DNET

A Communications Facility for Distributed Heterogeneous Computing

FINAL REPORT

Contract NAS 5 - 30085

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

DNET

INTRODUCTION GUIDE

Version: 1.24
Print Date: 09/01/89 13:14:37
Module Name: intro.gui

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

CONTENTS

1. Abstract 2
2. Organization of the Remainder of this Introductory Guide 4
3. Services Requested By NASA 5
4. Overall Topology of DNET 7
5. Defined Boundaries 8
6. Interfaces to DNET 9
7. DNET Implementation 10
8. Major DNET Components 11
9. Organization of the DNET Documentation 12

DAC Staff who contributed to DNET:

Principal Investigator
John Tole, Sc.D.

Engineers:

S. Nagappan
J. Clayton
P. Ruotolo
C. Williamson
H. Solow

Acknowledgements:

The DAC DNET project staff gratefully acknowledges the contributions of other persons and organizations to this effort.

- Barry Jacobs and Shyam Salona of the DAVID project at NASA Goddard Space Flight Center (NASA-GSFC) were the principal NASA contacts for this project. Their support and interfaces with other NASA personnel greatly facilitated all aspects of DAC's effort. Their understanding of the nature of this research also made the conduct of this project most pleasant for the project staff Their efforts are deeply appreciated.
- The DNET interfaces to TCP/IP and DECnet are based in part on the 'NET' code developed at Space Telescope Institute (STI). DNET also derives some of its design philosophy from the STI-NET system. DAC is grateful to STI for providing the NET code and for consultation on its use. Peter Shames and Steve Zeller of STI were especially helpful with information and access to STI resources.
- Todd Butler of the NSSDC facility at NASA-GSFC provided much useful consultation and assistance on DECNET.
- Jerome Bennett and Charles Cosner of the Data Flow Technology staff were unceasingly co-operative in resolving numerous questions and problems with the IAF and DFTNIC VAX machines at NASA-GSFC. This project could not have been completed without their assistance.
- David Pipes and Randy Thompson provided valuable administrative assistance on the **nssdcs** and **luesn1** SUN4 computers at NASA-GSFC.
- Bob Wood and Sally Saucedo of DAC provided ongoing system administration which is greatly appreciated.
- Chris Walters of Mitre Corporation offered important technical contributions on the configuration of an Ethernet System.

1. Abstract

This document describes DNET, a heterogeneous data communications networking facility. DNET allows programs operating on hosts on dissimilar networks to communicate with one another without concern for computer hardware, network protocol or operating system differences.

The overall DNET network is defined as the collection of host machines/networks on which the DNET software is operating. Each underlying network is considered a DNET "domain". Data communications service is provided between any two processes on any two hosts on any of the networks (domains) that may be reached via DNET. DNET provides protocol transparent, reliable, streaming data transmission between hosts (restricted, initially to DECnet and TCP/IP networks). DNET also provides variable length datagram service with optional return receipts.

Communications and computing services within DNET are provided in an environment based on clients and servers. When 'permanent' connections are required, clients request connections to specific servers by contacting a 'Master Server' at the destination host. The assignment of specific instances of server processes to clients is done by this Master Server as requests are received. The Master Server also controls server process creation, prespawning servers as necessary in order to improve response times. Local system administrators can regulate the number and type of specific servers. Servers report their status to this Master Server so it's database is always up to date.

Connectionless datagrams may also be sent between any two DNET processes. The DNET connectionless service is implemented separately from the streaming service.

There are two types of nodes in DNET, Hosts and Gateways. A DNET host is simply any machine which can access another machine via DNET. DNET gateways are special cases of DNET hosts which, provide protocol conversion "relays" between dissimilar networks in addition to other DNET functions.

DNET Host software includes a library of basic 'transport level' I/O functions, DNET application clients & servers, a master server (which controls the creation and allocation and permanent circuit connection of specific servers on its host) and a Datagram Master Server and Protocol Specific Datagram Servers which provide a universal interface to the DNET connectionless datagram service. DNET gateways include streaming 'relay' processes. These relays are simply special application servers which provide protocol conversion between the underlying networks. The DNET connectionless service handles protocol conversion between dissimilar networks as part of its inherent design.

Applications provided with DNET include File Transfer, Remote Login, and Remote Execution. In addition, a Network Command Interpreter allows I/O redirection and task 'chaining' across the network. Various application level processes may be invoked via this facility. DNET users may also add other applications by following interface techniques described in this document. Presentation level routines provide XDR data conversion capabilites in order to handle differences in internal data representation on different machines.

DNET also includes a provision for electronic mail and several network utilities of use in both system administration and user applications.

From the user's perspective, DNET is implemented as a library of program callable functions with input, output, and error redirection capabilities. No Kernel Modifications are required on any machine on which the DNET software operates. While this constraint introduces some potential performance problems, it greatly simplifies the logistics of implementing and maintaining a heterogeneous network.

While DNET has been designed for initial use by the NASA-GSFC DAVID project, consideration has also be given to its future utilization with other applications which must operate in a heterogeneous environment. The initial DNET environment is thus limited to TCP/IP, DECNET, and dialup communications alternatives and UNIX (ATT System V and BSD) and DEC VMS operating systems. Design generality has been maintained as much as possible however, so future inclusion of other operating systems and communications facilities, especially ISO/OSI, UNIX/uucp, and IBM SNA/LU6.2 and VM/CMS may be contemplated.

2. Organization of the Remainder of this Introductory Guide

The remainder of this guide contains the following information:

- Discussion of Networking Services Originally requested by NASA
- Overall Topology of DNET
- Current Defined Boundaries
- Definition of Interfaces to DNET
- Overview of Major DNET Components
- Implementation
- Introduction to Documentation Organization

3. Services Requested By NASA

The following is a brief description of the communications services which NASA required for the DAVID project.

Task To Task Communications

1. Initiation of program task at a remote internet (DAVID) node with facility to pass arguments/results and propagate termination signals.
2. Transfer and execution of portable programs at an internet (DAVID) node
3. Provide transparent operations which allows 2 programs or command procedures running on different DAVID nodes on different host environments to communicate with one another.
   Operations would include:
   - initiation of remote tasks
   - termination of remote tasks
   - send data & 'interrupt' messages
   - receive data & 'interrupt' messages

File Transfer

1. File transfer of ASCII and binary files to any internet (DAVID) node with multiple authentication options including:
   - autologin
   - various user/passwords
2. file transfer between any two internet nodes, neither of which is local to the user
3. end to end reliability with timeout and acknowledge options

   NOTE: the ability to specify some or all timeout parameters may be dictated by underlying protocols and not under control of software DAC is able to provide.
4. Provides presentation layer function for data conversions so as to make differences in data type representation transparent across machines.
5. Provision for initiation of remote procedure upon successful completion of file transfer
6. Additional operations
   - check for existence of file
   - delete, rename, append to file

Remote Login

1. Supports internet logons (with relay mechanism)

2. Supports different authentication methods
    - autologin
    - username - password
3. Supports different terminal types
4. Provides option to specify execution of user defined logon procedure

General Utilities

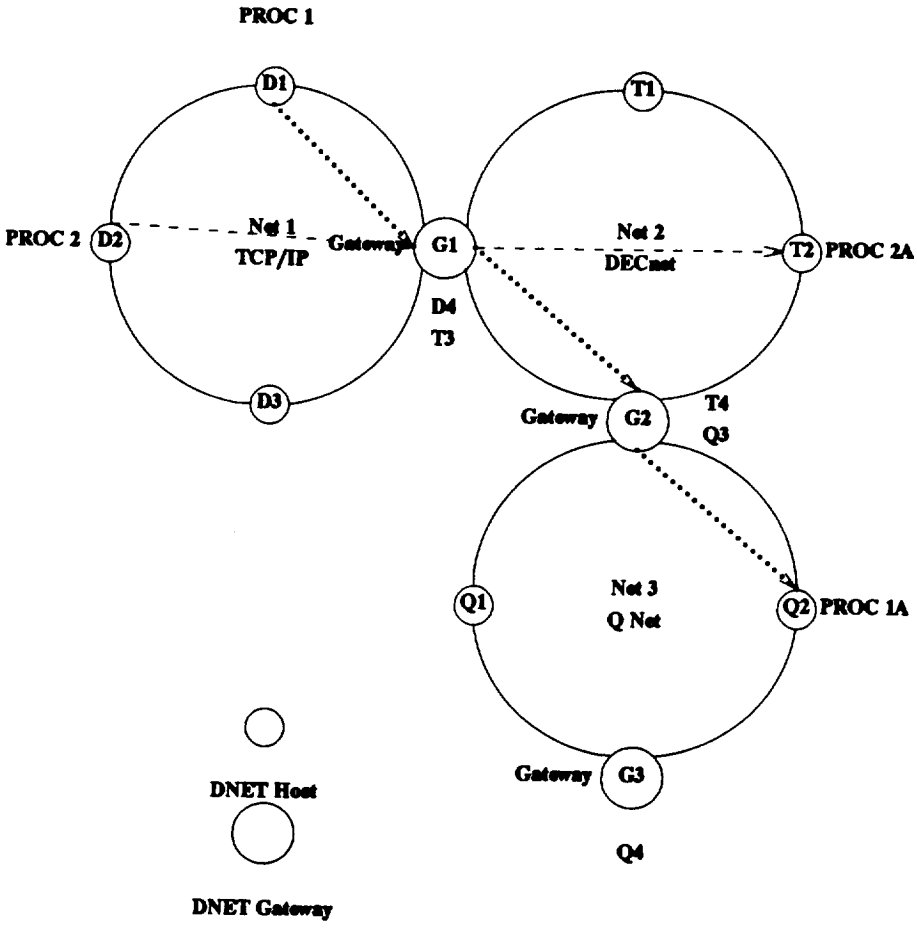
1. Indication that remote internet node is up
2. Ability to determine load on remote node
3. Utility to determine host ids, host names, and host aliases

Mail

1. Provide capability to send mail to one or more people or tasks at various internet nodes

4. Overall Topology of DNET

The overall topology of DNET is a collection networks as shown in the following diagram:



Each of the networks contains one or more nodes (host computers) and some of these nodes (gateways) are shared by two or more networks. From the perspective of a DNET user, all of the networks and nodes appear identical. Positive identification of a specific node requires only that the name of the destination node and the network on which it resides be known. No knowledge of the path between, nor the environment under which the destination node operates is normally required. This, of course, differs radically from the view that DNET implementors and administrators see. The latter view involves a collection of incompatible networks and environments that must be combined to bring reality to the prior view. In order to provide for such a reality, certain boundaries must be defined as described in the next section.

5. Defined Boundaries

The following 'Boundaries' exist for DNET. The boundaries are simply a compact list of the environments in which DNET can be expected to operate without an excessive software porting effort. The current boundaries are:

1. Communication Protocols
   - TCP/IP (Wollongong, Excelan, and Berkeley Implementations)
   - DECnet
   - BSD Sockets
2. Operating Systems
   - UNIX (System V.2 and 4.2BSD)
   - VMS
   - MS-DOS (DNET Clients Onlyd)
3. Hardware
   - AT&T 3B2
   - Sun Model 3
   - DEC VAX
   - IBM PCs (DNET Clients Only; LAN interface only)

Despite support for a variety of environments and underlying components, the interface to DNET users will remain standard.

6. Interfaces to DNET

There are three interfaces defined for the DNET network::

- End User
- Programmer
- Administrator

The end user is a person who takes advantage of the networks services through an interactive mode involving utilities that are run from the keyboard. These generally manifest themselves as interactive distributed services like trivial file transfer protocol, electronic mail, and remote command language. The user sees DNET as a 'homogeneous' network. All commands to operate applications and the behavior of these applications appears to be uniform across all machines on the DNET network.

The interface provided to the programmer comes in two basic forms: the connection oriented services, and the connectionless oriented services. The connection oriented services provide a streaming mode of full duplex conversation between two processes. The connectionless mode services provide a method of sending and optionally receiving datagrams (packets of information) without previously establishing a connection. Both of these services are implemented through user libraries that may be compiled into the programmer's applications. The DNET user is provided with extensive documentation to facilitate usage of the system.

The system administrator is provided with utilities for starting and stopping a DNET node, modifying the number and types of DNET application servers at the node, altering routing tables and monitoring the status of both local and remote DNET nodes.

For more advanced applications, information is also provided on DNET 'internals' to allow more sophisticated special services to be implemented. This information is provided in the DNET technical guide and reference.

7. DNET Implementation

The implementation of DNET was designed with low impact on target machines as a high priority. This low impact philosophy is intended to apply not only to resource consumption on the local machine, but also to administrative and user functions. From a resource standpoint, DNET daemon processes only require CPU resources when applications request service, and the resources provided by the underlying networks are still available to programs that have already been written to interact directly with them. All administration tasks associated with the underlying networks remain the same, and the administrator's responsibilities are clearly outlined in the administrative guide. The end user utilities were created with preexisting standards so that retraining is minimized, and the programmer tools were kept to a minimum, are well documented, and many times operate in a similar fashion to standard file operations.

The DNET design also takes into account the importance of simplicity in adding new applications in a heterogeneous environment. DNET specifies a minimal set of rules for writing client-server pairs to implement new applications. Details are provided in the DNET Programmer's Guide and Reference.

8. Major DNET Components

A small collection of DNET components are required on each node on the DNET network:

DNET Master Server

This server provides a 'well known' port for the connection oriented DNET services. All server applications (see Programmer's Guide) are started and maintained by the master server. One master server is required per protocol per node. Thus, a DNET node which is connected to both DECnet anda TCP/IP will have a Master Server 'listening' on each of these two networks.

DNET DataGram Master Server (DGMS)

The DNET DGMS is the heart of the connectionless service and provides all routing and switching services for datagrams either coming in from a network, or from it's genesis in a user process.

per protocol DataGram Server (DGS)

This component provides the low level interaction with a particular underlying provider (ie TCP/IP). One such server is required for each protocol at a particular DNET node. All DGS components then provide a standard interface to interact with the DGMS component so that all networks appear to have the same interface. Beyond that, the DGS components are merely dumb relays.

Relay processes

These processes provide relay service for the connection oriented service and are only found on gateway machines. The relay processes actually write out on the proper network in loopback mode to get to the master server controlling that network type.

Application Processes

These processes provide the 'standard' collection of DNET applications for the user.

Administrative Processes

These are a small collection of scripts and rules which allow the local system administrator to control the local DNET functions.

User Library

The user library contains all of the routines necessary for a programmer to use the DNET services. A separate set of routines are provided for connection and connectionless oriented services.

9. Organization of the DNET Documentation

The DNET documentation is organized around the three interfaces (user, programmer, administrator) defined above. These documents together with this Introduction Guide and a Technical Guide describing the internal implementation details, provide a complete description of DNET. There are thus documentation categories for end users, programmers, network administrators, and internal programmers. Each of the above mentioned categories is divided into two manuals, a guide and a reference. Providing two manuals per category allows the documentation to act as both a quick reference for users who need only specific details, and as a learning or refresher guide. Utilities are provided so that the reference manuals may actually be stored on-line if space allows so that a DNET user may interactively reference the manual from their terminal. Additional Notes are provided for each category, as appropriate.

DNET

USER'S GUIDE

Version: 1.21
Print Date: 09/01/89 13:43:05
Module Name: user.gui

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

CONTENTS

1. DNET Overview 1
   1.1 What is DNET? 1
   1.2 Major Elements of a DNET Network 2
      1.2.1 Network Arrangement 2
      1.2.2 Existing Networks 3
      1.2.3 DNET Hosts 3
      1.2.4 Gateways 3
      1.2.5 DNET Routing 3
   1.3 What the DNET User Has Available 4
      1.3.1 Applications 4
      1.3.2 Presentation Level Services 4
      1.3.3 Basic I/O C Function Library 4

2. How to become a DNET User 5

3. An Introduction to DNET Applications 6
   3.1 The Echo Routine 6

4. Help Facilities 8
   4.1 Manual Pages 8
   4.2 On-line Help 8
      4.2.1 UNIX 8
      4.2.2 VMS 8

5. File Transfer 9
   5.1 DNET TFTP - DNET Trivial File Transfer Protocol 9
   5.2 Invoking DNET File Transfer 9
      5.2.1 DTFTP from the Command Line 9
   5.3 User Commands 10
   5.4 File Transfer Errors 11
   5.5 A File Transfer Example 11
   5.6 Using the Network Command Language for File Transfer 12

6. Remote Login 13
   6.1 Introduction 13
   6.2 Invoking DNET Remote Login 13
   6.3 Ending the Remote Login Session 14
   6.4 Security Issues 14
      6.4.1 UNIX 14
      6.4.2 VMS 14

7. Network Command Language 15
   7.1 Overview 15
   7.2 Network Command Processor Schematic 15
   7.3 Network Command Language 15
      7.3.1 Command Language Syntax 16
      7.3.2 Using The Command Language 16
   7.4 Network Command Interpreter 16
      7.4.1 Status Reporting (from last Network Command Server) 17
   7.5 Initiation of File Transfer from One Remote Node to Another 17

8. Remote Execution 19
   8.1 Passing Arguments to Tasks 19
   8.2 Transfer & Execution of Portable Programs at a Remote Host 19
   8.3 Initiation of Remote Procedure Upon Completion of File Transfer 19

9. Electronic Mail 20
9.1 Mail Operation 20
9.1.1 Sending Mail 20
9.1.2 Reading Mail 21
9.1.3 Auto Notification of Mail Arrival 21
10. General Network Utilities - dnetstat 22
10.1 Testing if DNET is alive 22
10.2 Obtaining Status of DNET Servers 22
11. Presentation Services 25
11.1 XDR Services 25
12. Glossary 26

1. DNET Overview

This section provides an introduction, from the user's perspective, to the DNET Network for Heterogeneous Distributed Communications. The various functional elements which make up DNET are described as are some of the important assumptions made in the design.

1.1 What is DNET?

DNET is a communications environment which provides a consistent view of a number of interconnected heterogeneous networks. Networks included at present are those which use the DECnet and TCP/IP communication protocols. DNET provides a 'seamless' or uniform interface to these networks and machines, giving the impression that a single homogeneous network is being used.

The basic DNET software provides a consistent 'Transport' Level interface to the underlying existing networks/protocols on which in operates. Various applications may use these transport facilities for their communication needs. DNET includes a set of commonly used 'generic' applications as a basic working set of tools and as examples of how this communication facility may be used.

NOTE: DNET (currently) operates as an application on machines on which it is available. While this implementation strategy introduces some potential performance problems, it greatly simplifies various logistical problems in operating a heterogeneous network. Further rationale for this approach is provided in the DNET Technical Guide.

1.2 Major Elements of a DNET Network

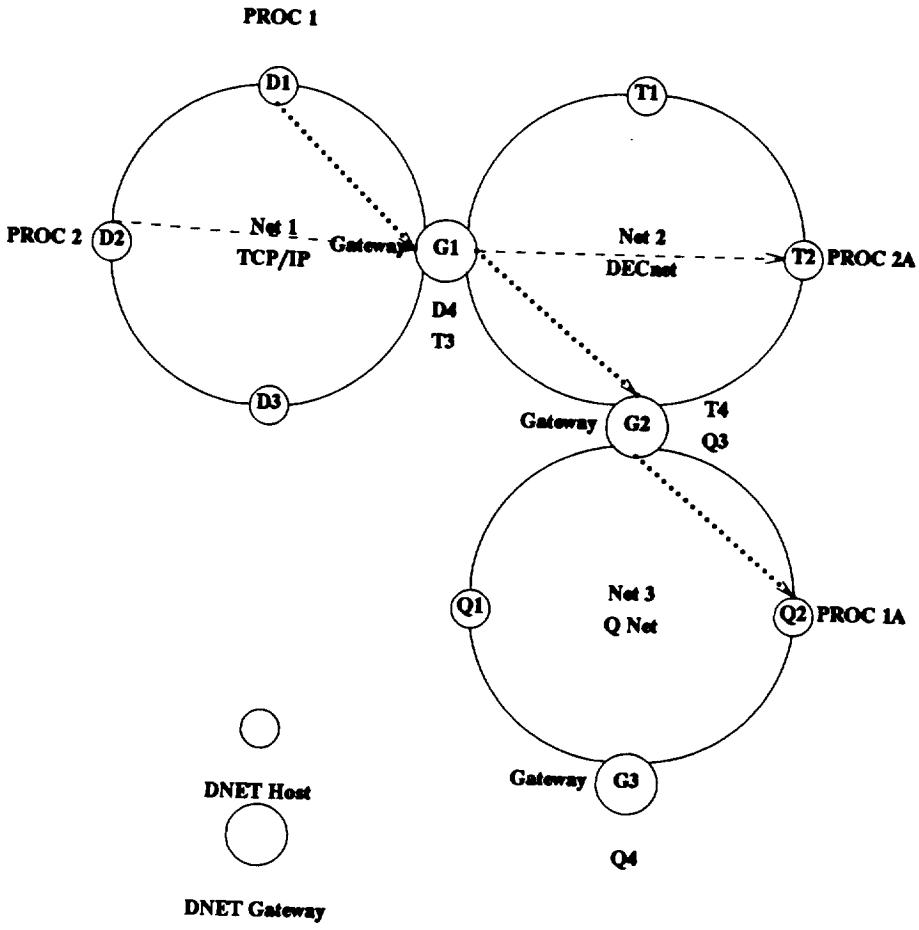
A DNET network consists of the following major elements:

1. A collection of two or more existing, specific networks (with protocols supported by DNET, currently TCP/IP, DECNET, and Asynch Dial-up)
2. DNET Hosts - machines which are able to communicate using DNET services
3. DNET Gateways - special DNET Hosts which also provide protocol conversion between the underlying networks

By implication, the DNET Hosts and Gateways have DNET software installed which establishes their functions. Each of these elements is described in more detail below:

1.2.1 Network Arrangement

DNET is a "meta-network" or a network of networks. The general arrangement of these major elements of a DNET network is shown in the following diagram.



DNET can establish a "permanent virtual circuit". In this mode an "open" function is called to establish a communications path from one process to another process in another host, possibly in another network. The path established comprises relay processes and network connections dedicated exclusively to the stream mode transport of data between the end points of the circuit. Permanent virtual circuits reduce the number of network connections that must be established and the associated task initiation required. This significantly improves network performance. When data is transmitted in a "streaming" fashion in one session the performance increase more than offsets the initial cost of circuit establishment.

DNET also provides variable length datagram service. The user interface to this service is connectionless (i.e. no "open" is required before starting process to process communications). Datagrams may be used to either transmit data or signal information.

1.2.2 Existing Networks

The underlying networks associated with DNET are ones which have existing reliable, data streaming capabilities. The networks in which DNET may currently operate are TCP/IP, DECnet, and Asynchronous links. DNET depends on the transport services of these these 'underlying' networks and presumes that they are operational.

1.2.3 DNET Hosts

DNET Hosts are computers at which local processes may use the facilities of DNET to interact with remote processes in the heterogeneous network. Any computer connected to one of the networks served by DNET may become a node on DNET provided the following conditions apply. The machine must:

1. be resident on a specific existing network (e.g. TCP/IP Net X, SPANET, etc.) which is known to DNET
2. have at least one DNET master server listening on a known DNET network. This requires the following processes: The DNET PVC Master Servers(s) **DMSDEC and/or DMSTCP**, and the Datagram Master Server(s), **DGMS** and Datagram Protocol-specific servers: **DGSUDP and/or DGSDEC.**
3. have at least one DNET application server running (if requests from remote nodes are to be serviced)

1.2.4 Gateways

DNET Gateways are nodes in DNET which are connected to one or more networks in which DNET is operating. The function of the gateway is to bridge the protocol and other differences between these networks in a transparent manner. The gateway functions are implemented in special DNET PVC Relay servers and Datagram Servers which provide protocol conversion for Permanent Virtual Circuits and connectionless datagrams respectively. Except for their special-purpose function, these servers are handled just like any other DNET application servers.

1.2.5 DNET Routing

DNET employs hierarchecal routing. Each DNET node contains a routing table which indicates, for each network known to DNET, the next host to contact in which to 'move' toward that particular network. In general, the next hops listed in the table are all DNET gateway machines. The user generally need not be concerned with the routing tables, however a 'map' of the DNET network or at

least the names of remote DNET nodes and networks of interest is essential for use of DNET facilities.

1.3 What the DNET User Has Available

1.3.1 Applications

Certain common application facilities are provided with DNET. Each of these applications has a corresponding application server within the domain in which it is available. Operation of specific applications will be at the discretion of the administrator at the destination node. The DNET function **dnetstat**, described elsewhere may be used to determine which servers are running at a specific DNET node.

The available applications include:

1. File Transfer - loosedly based on TFTP with some enhanced features
2. Remote Login - similar to 'telnet' or 'set host' - allows full interactive sessions with UNIX servers and more limited sessions with VMS servers
3. DNET Network Command Interpreter - A generalized remote execution and task redirection application - Similar to the redirection capability of UNIX
4. Mail - a basic system similar to a stripped-down UNIX mail
5. DNET Status - a generalized network status utility similar to 'netstat'

1.3.2 Presentation Level Services

DNET provides a limited Presentation Level Service for use by the above applications (and user definined applications) This service allows:

- Conversion of Data Elements between dissimilar machines via the SUN XDR (External Data Representation) functions.

1.3.3 Basic I/O C Function Library

The DNET Basic I/O functions may be used to generate custom operations in the DNET environment. The I/O functions provide communications facilities between tasks on different hosts within DNET. These facilities include permanent virtual circuits, connectionless datagrams, and signalling.

The I/O library may be used in two ways:

1. **Low Level Connectionless Task-to-Task Communications** - using the Datagram and Signalling functions contained in the I/O package, the user may communicate with other processes elsewhere in DNET
2. **User Specific Custom Applications** - by following the conventions for DNET Client-Server relationships, the user may write higher level custom applications which will operate smoothly in the heterogeneous DNET environment.

Further discussion of the basic I/O package is found in the DNET Programmer's Guide and Reference Manual.

2. How to become a DNET User

1. The machine which you are using must be 'known' to DNET and have DNET running locally.
2. The path to DNET 'client' files must be known to your account. This information needs to be placed in the appropriate file as noted below:
   1. UNIX

      .profile

      set dnet_home and PATH to ../dnet/bin

   2. VAX/VMS

      The dnet_home directory must contain the following machine specific file:

      DAC Microvax II
      dnlogin.dv

      NASA-GSFC - DFTNIC VAX
      dnlogin.dft

      The file

      login.com

      in the user's login directory must have the following lines. The example given is for the DAC Microvax II machine. The definition for **dnet_home** and **dnlogin.XXX** will be machine dependent.

      ```
      $! DNET Specific Environment
      $ set proc/priv=grpnam
      $ define/group dnet_home $disk1:[sys0.dnet.dnet]
      $! run DNET login script
      $ @dnet_home:dnlogin.dv
      ```

   The DNET System Administrator's Guide provides details on information to be included in these files as well as other useful information about DNET configuration.
3. You must be aware of network and host names for the machine(s) with which you wish to interact.
4. You need to become familiar with the DNET applications which you need to use. These are presented in the following sections.

3. An Introduction to DNET Applications

This section provides a brief introduction to the use of typical DNET applications. The several applications have been designed and implemented with two purposes:

1. To make DNET immediately useful in solving typical user problems, even with networks of limited scope.
2. To serve as examples of how to use the DNET tools to build other applications.

DNET Applications operate in a consistent manner at all nodes defined within the DNET network. In order to use DNET applications, the user must only be aware of the DNET destination host and destination names and the specifics of the application. A typical DNET application is invoked with the command line sequence:

Client-command Destination-network Destination-host

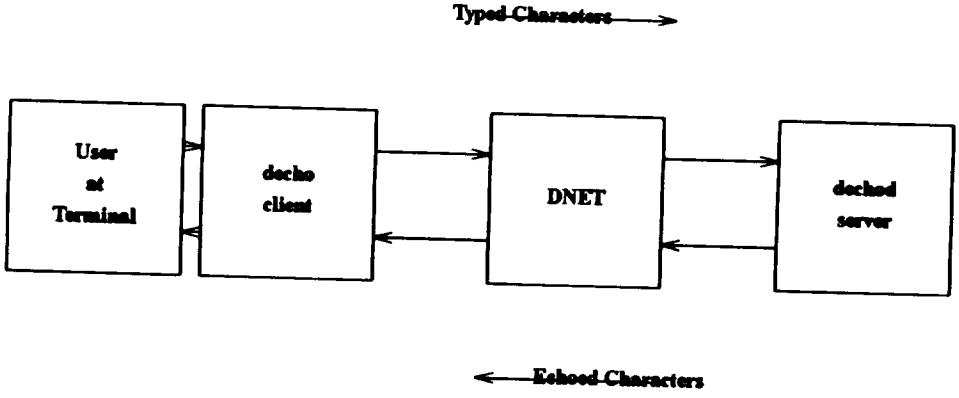
For example, if the user is located at some machine on DNET and wishes to perform a file transfer to/from the IAF VAX at NASA-GSFC, he would enter the following command to start the DNET basic (trivial) file transfer application.

dtftp spanet iaf

Other applications are invoked via a similar syntax.

3.1 The Echo Routine

An elementary echo routine, **decho**, provides a convenient introduction to the use of DNET applications. **decho** allows the user to enter lines of arbitrary text which are then echoed back to the terminal from a remote echo server. Experimentation with this simple function will give the user a feel for the typical setup time for DNET streaming connection and an introduction to error conditions and run-time debugging options which are available to the user.



decho is invoked by entering the following command line:

decho dest_network dest_host [CR]

A message will be printed on the terminal:

Attempting to connect to dest_network dest_host

In 2-10 seconds, the connection to the destination should occur. **decho** should then respond with either:

Ready

or:

dechod server unavailable at destination (DNET error xxxx)

Assuming the 'Ready' prompt appears, one may then type an arbitrary line of text, e.g. :

12345 [CR]

After a short delay, this text should appear on the screen a second time as it is echoed from the remote server. This process may be continued indefinitely. The 'echo' delay provides an instantaneous, subjective indicator of the performance of DNET. Keep in mind that this delay is heavily dependent on the load on intermediate DNET nodes which may be performing protocol relay operations.

When finished using decho enter the following, machine dependent terminator to exit to the operating system.

UNIX:

Cntrl D

VAX/VMS:

Cntrl Z

4. Help Facilities

There are two sources of help for DNET applications, manual pages in the DNET User's Reference found with this documentation, and on-line versions of the same manual pages on machines where this facility is supported.

4.1 Manual Pages

Manual pages in the User's Reference follow the style of UNIX manuals. The user is referred to examples in this Reference.

4.2 On-line Help

DNET provides an online 'manual' facility which may be of help to the user.

The manual pages found in the various DNET Reference Manuals are available on-line as follows:

4.2.1 UNIX

The DNET manual may be invoked by entering:

dman dnet_function_name

where dnet_function_name is the DNET function or application for which additional information is desired.

4.2.2 VMS

On-line Help for VMS has not been implemented in this release

5. File Transfer

5.1 DNET TFTP - DNET Trivial File Transfer Protocol

The DNET File Transfer protocol is loosely based on TFTP (Trivial File Transfer Protocol) based on the Network Working Group RFC 783 dated June 1981. This standard was selected because it is simple, yet capable of transfering files in the DNET environment. DNET provides the reliable streaming transport facilities, while the TFTP protocol provides the command processing, data formating, error detection, reporting and recovery.

Both binary and text files may be transmitted with **dtftp**. **Dtftp** also has a number of features beyond those provided by TFTP including local and remote directory, directory listing, and change directory and a user warning against inadvertant overwriting of an existing file.

dtftp also requires a username and passwd to be entered in order to connect to the remote system.

5.2 Invoking DNET File Transfer

5.2.1 DTFTP from the Command Line

The DNET file transfer facility is invoked at the command line by entering:

dtftp [dnet_network] [dnet_host] [CR]

If the network and host are specified, DNET will immediately attempt to contact the file transfer server at that location. If the destination is not specified, dtftp will start up in a local or disconnected mode.

This will cause the file transfer prompt to be displayed:

dtftp>

If the destination was not specified on the command line, a connection may now be attempted, if desired by entering the command:

connect

The connect request will require the network and host information. You will be queried for:

Network:

and the

Host:

Then the following message will appear:

Attempting to connect to [network] [host]

Once a connection has been established with the remote host, you will be prompted for a login account at that machine:

Login:

Enter a valid account name for the destination host.

You will then be prompted for a password associated with this account:

Password:

Enter the password for the account name just entered.

When the account information has been verified, the client will respond with the message:

CONNECTED

You may now proceed with other commands as discussed below:

5.3 User Commands

The following commands are provided with **dtftp**. These commands are self-explanatory except as noted.

cd XXX - change the default directory on the remote host to XXX

get name [newname *] - retrieve a file from the remote to the local host

help - display help message for available dtftp commands

lcd XXX - change the default directory on the local host to XXX

lpwd - list the current directory on the local host

ls - list the contents of the current directory on the remote host

lls - list the contents of the current directory on the local host

! command string - Allows execution of a local command

mode - Allows specification of binary or ASCII mode

put name [newname *] - transmit a file from the local to the remote host

pwd - list the current directory on the remote host

quit - end the file transfer session

*** - in get or put operations, if newname is not given, it is assumed that name is the target file.**

If the target file already exists, a warning message is presented:

Destination File Exists - Overwrite (y/n)?

If you answer yes (y), the old file will be overwritten. (In VMS, this will not actually occur as a new file extension will be assigned to the target file, however the warning message is consistent for both UNIX and VMS).

5.4 File Transfer Errors

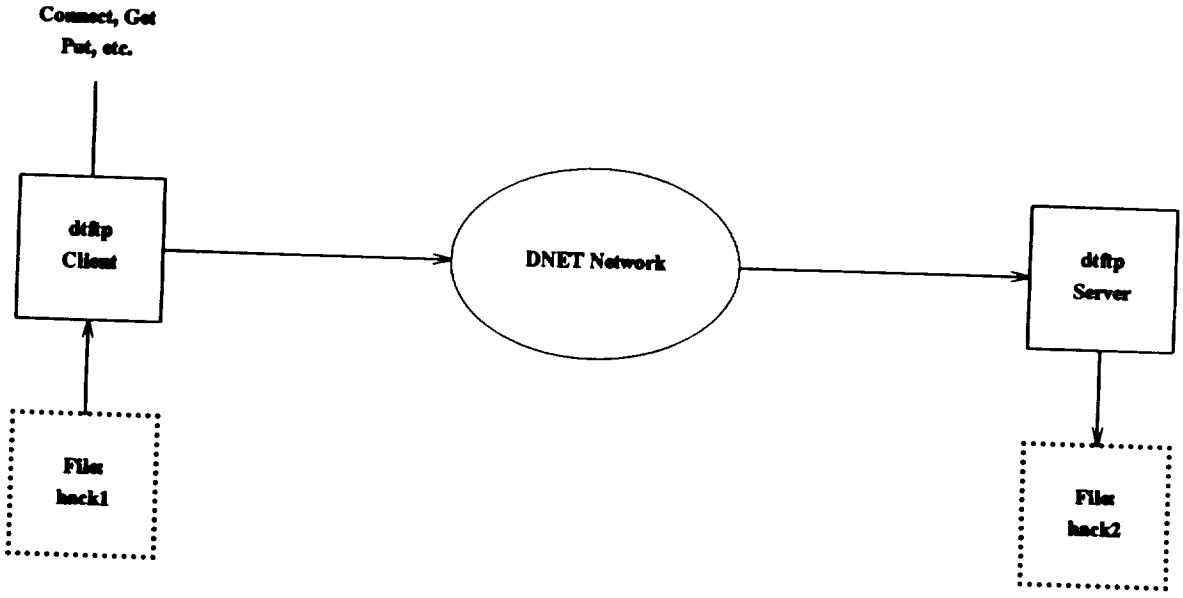
Error reporting from DTFTP includes the following:

1. Login incorrect
2. File Not Found
3. File I/O Error
4. File Access Violations

Except for the obvious question of access privileges associated with login failure, most non-fatal 'failures' are self explanatory.

5.5 A File Transfer Example

As an example of the use of **dtftp**, consider the following diagram:



We wish to send the file **hack1** on the Client Machine to the Server Machine, renaming it in the process to **hack2**. The series of commands used to perform this task are shown below:

```
$ dtftp spanet dacvax
attempting to connect to spanet dacvax
Login: dnet
Password: ******
CONNECTED
dtftp> put hack1 hack2
completed ascii put of hack1 to hack2
dtftp> quit
$
```

5.6 Using the Network Command Language for File Transfer

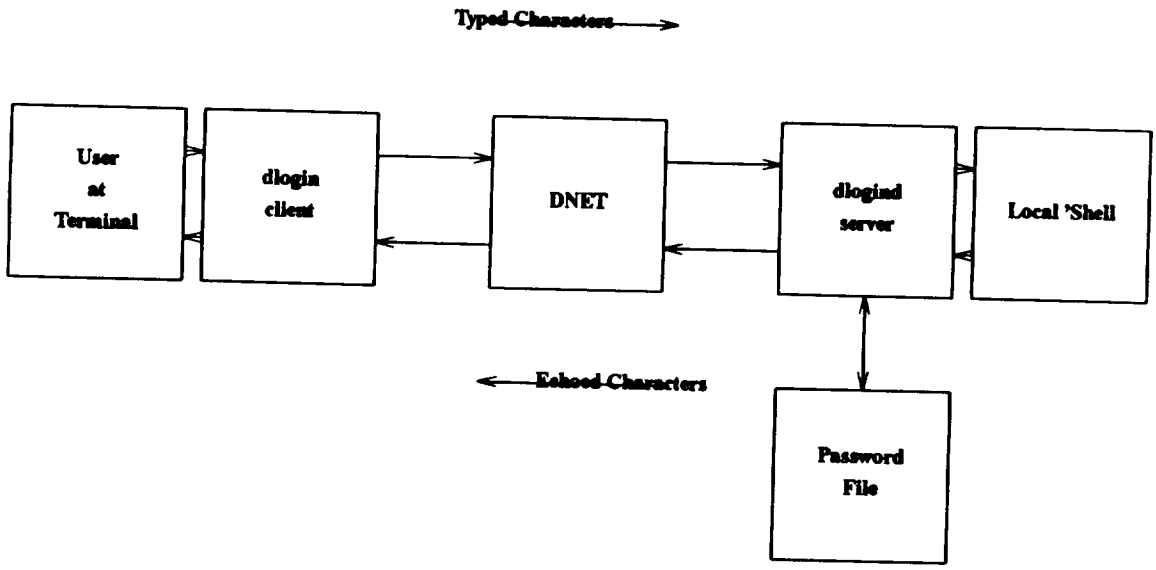
File Transfer may also be accomplished using the DNET Network Command Language (NCL). There are advantages and disadvantages to using the NCL for file transfers. This option is discussed in the Network Command Language Section of this Guide.

6. Remote Login

6.1 Introduction

The DNET remote login application allows the user to log onto and carry on an interactive session with a distant DNET host. Once connection has been made to the remote host, the user will appear to be directly connected to that host. Only instantaneous network performance should affect the ability to do work, including screen oriented editing. Thus **dlogin** is similar to the telnet, rlogin, or set host xxx facilities with which the user may already be familiar.

A schematic of the DNET remote login is shown in the diagram below:



Currently, DNET remote login may be initiated from either UNIX or VMS hosts, however the remote host must be a UNIX machine in order for complete interactive operations to take place. (An interactive VMS DCL 'shell' has not been implemented to date under DNET).

6.2 Invoking DNET Remote Login

The DNET remote login facility is invoked at the command line by entering:

dlogin [dnet_network] [dnet_host] [CR]

Once a datastream has been opened to the destination, you will be prompted for account information:

login: xxxx
Password: *****

If this information is determined to be correct by the remote machine you will placed in your 'home' directory and your preferred 'shell' is executed. You may now perform any interactive operations which you might do were you directly connected to the remote system.

6.3 Ending the Remote Login Session

When the remote login session is complete, simply enter **cntrl-D** (or exit or logout, if csh) to abort the session and to close the DNET virtual circuit between your local machine and the remote host.

6.4 Security Issues

When the client invokes Remote Login, authentication of the client is done by the login process at the remote host. Subsequent process spawning and/or remote login to other hosts from processes created by the initial client will all carry the access rights permitted to the initial client.

6.4.1 UNIX

At UNIX servers, the file **/etc/passwd**, controls access to the system. This file is consulted by the **dlogin** server in order to validate a user at this host.

6.4.2 VMS

Currently, due to problems accessing the UAF file on VAXes, the password is hard-coded for VMS servers, an obvious security problem for a production network. This limitation must be corrected before DNET is used in a production environment.

7. Network Command Language

7.1 Overview

The DNET Network Command Processor is a command language processor for use in a heterogeneous multi-network environment. This DNET facility allows very general control of processes across the heterogeneous network and provides for redirection of input/output streams between files and/or processes located at arbitrary DNET Hosts.

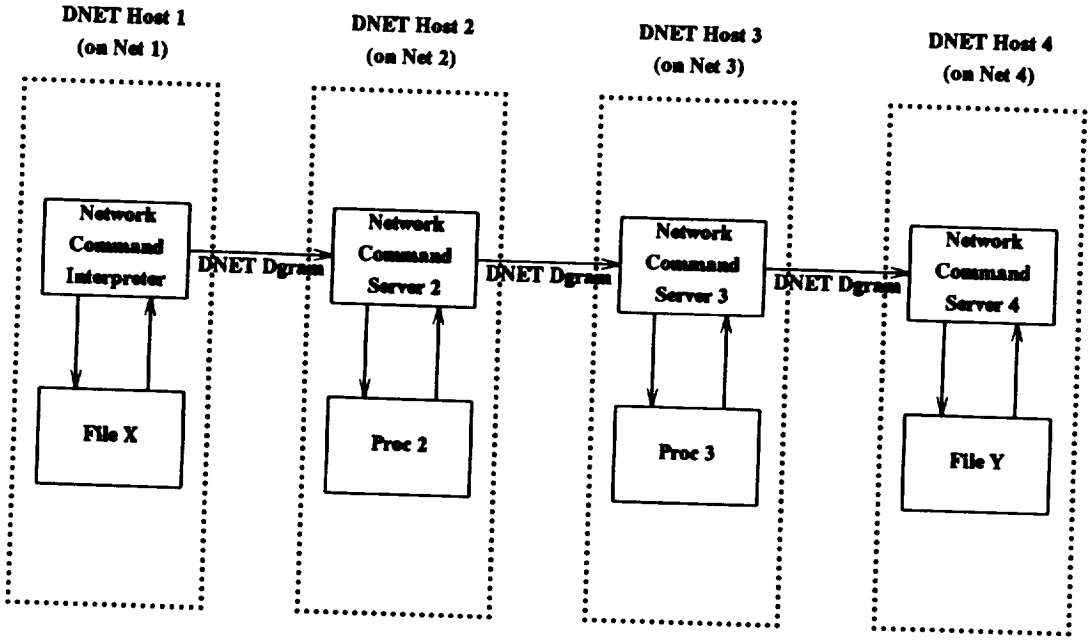
Three Software Elements are required for the Network Command Processor.

- Network Command (Client) Interpreter - used at the initiating node to interpret the Network Command, provide any local service requested, including process spawning, and send remainder of command to next net/host/process in the command chain
- Network Command Server - services network command which arrives from NCI or another NCS; provides any local service requested, including process spawning, and sends remainder of command to next net/host/process in the command chain The NCS is a DNET application server and is thus registered in the PVC Master server tables.
- Network Command I/O Process - special 'cat' equivalent process which may be spawned by the NCI or NCS to provide streamed I/O to/from files.

7.2 Network Command Processor Schematic

A Schematic view of the relationship between these components is shown in the figure below. The generic command string being executed is:

Net1::Host1:File X > Net2::Host2:Proc2(param1,param2) > Net3::Host3:Proc3 > Net4::Host4:File Y



7.3 Network Command Language

7.3.1 Command Language Syntax

There are two types of objects used on the command line - Files and Tasks. Output can be directed to either a file or another task. The command language makes a distinction between files and tasks (executables) by preceding tasks with an "*".

Filename syntax is: network_name::host_name:filename

Taskname syntax is: network_name::host_name:*taskname(param1, param2 ...)

Initially, there is only one network command language operators, ">". The ">" indicates redirection of standard output.

An example command is:

starnet::xhost:cfile > yhost:*sort > myfile

This NCL command will send the file named "cfile" from the host "xhost" to the host "yhost". On host "yhost" the cfile will be sorted using the "yhost" resident sort utility. The resulting output will then be saved in the file "myfile" on the local system.

Other examples are given below.

When the network name or host name is not specified the local name is assumed. Spaces around the ">" are optional.

7.3.2 Using The Command Language

When filenames appear in command strings they imply the execution of file i/o servers. The network command:

dac_net::vax2:david_comm > g_net::host1:*checkp > results

requests that the contents of a file "david.comm" on host "vax2" in the network "dac_net" be input to the task "checkp" executed on "host1" in the network "g_net, and the output be placed in the file "results" in the host on which this network command is being executed.

The network command:

net_one::vax6:c-file > host1:s-cfile

requests that the contents of a file "c-file" on host "vax6" in the network "net_one" be copied to the file "s-cfile"

7.4 Network Command Interpreter

The Network Command Interpreter (or client) is invoked as an application from the shell prompt on the local system.

%) dncl

dncl > command_string1

response to CS1

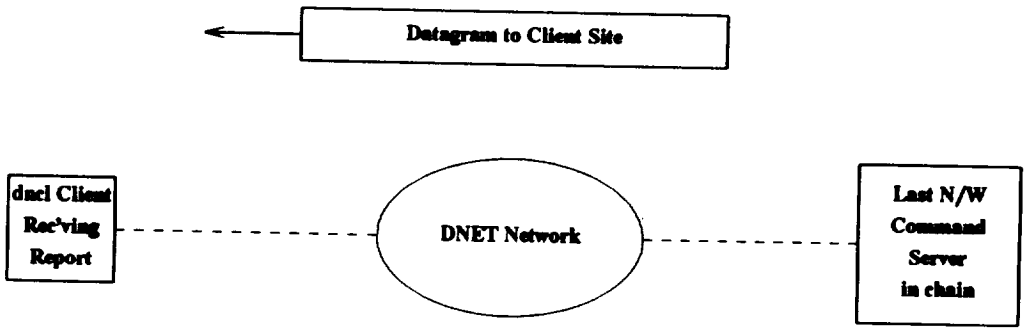
dncl > command_string2

response

etc...

7.4.1 Status Reporting (from last Network Command Server)

When the command line specified by **dncl** has been executed, an acknowledgement is propagated to the initiating client process as shown in the following diagram:



7.5 Initiation of File Transfer from One Remote Node to Another

The Network Command Language may be used at a third party location to initiate file transfer. A typical command would be:

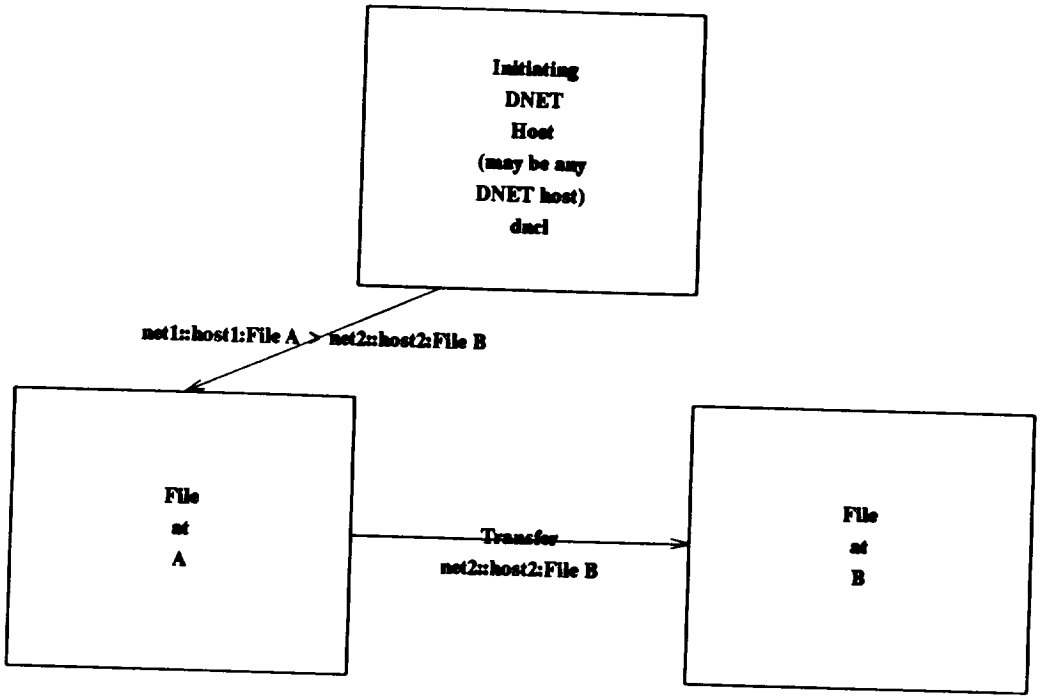
dncl > net10::host3:filexx > c-net::fhost:newfile

or

dncl > mynet::host6:*dtftp filename options > newfile

Where filename and options are parameters to the file transfer task "**dtftp**".

The effect of such a command is shown in the following diagram:


Initiating
DNET
Host
(may be any
DNET host)
dncl
net1::host1:File A > net2::host2:File B
File
at
A
Transfer
net2::host2:File B
File
at
B

8. Remote Execution

8.1 Passing Arguments to Tasks

Passing arguments from a calling task to Network Command Processor is by command string, as described in the preceding section. This is used in both the terminal interactive and the "C" language interface.

8.2 Transfer & Execution of Portable Programs at a Remote Host

File transfer and execution is implemented using the network command processor. For example the commands:

dncl > net3::host5:filez > net3::host2:workfile

dncl > net3::host2:*workfile

will transfer and execute a file.

8.3 Initiation of Remote Procedure Upon Completion of File Transfer

It is also possible to use the DNET Network Command Language to perform a file transfer followed by the execution of a remote procedure. Several alternatives are possible.

1. Two separate commands:

   transfer the file

   dncl > bnet::host3:file4 > c-net::xhost:newfile

   followed by

   execute the remote procedure

   dncl > c-net::xhost:*format newfile

2. **One 'composite' command:**

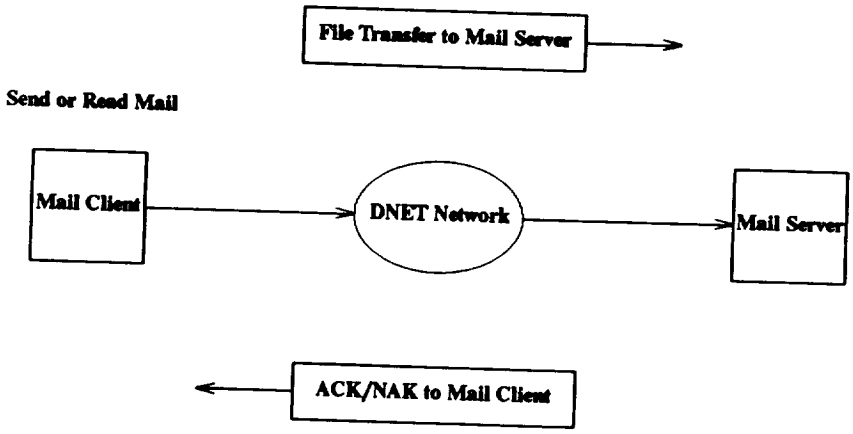
   dncl > bnet::host3:file4 > c-net::xhost:newfile > c-net::xhost:*format

9. Electronic Mail

DNET provides a basic Electronic Mail facility. This facility allows mail to be sent from the local DNET hosts to known users at other DNET hosts. Mail which has been received from other hosts may be read at the local host.

9.1 Mail Operation

The general operation of DNET mail is shown in the following diagram:



9.1.1 Sending Mail

DNET mail is invoked using syntax similar to that of other DNET applications as indicated below.

dmail network host user

where network = DNET network

host = hostname on that network

user is presumed to be a valid user on the destination machine

The sender then responds to the following prompts

To: *receiver_login_name_at_destination*
From: Senders name
Subject: xxxx
Cc: xxxx

Please enter short message and end with three CRs

this is a test
CR
CR
CR

The following prompts will then appear as the message is sent

CONNECTING
Completed ascii put of mail to destination

9.1.2 Reading Mail

To check for any DNET mail which may have arrived at your location simply enter the command

dmail

and request the Read Mail Option.

9.1.3 Auto Notification of Mail Arrival

DNET Mail includes a provision for automatic notification of mail arrival each time a user logs in. If mail is present for your account, the message:

You have DNET mail ....

will be presented as part of your login process.

NOTE: in order for this feature to be activated, the appropriate DNET login script must be part of your .login, .profile, or login.com file. This file must cause the file 'checkdmail' to be executed as part of the login process.

10. General Network Utilities - dnetstat

DNET provides a general network utility function **dnetstat** which allows the user to determine a variety of information about local or remote DNET nodes. Information which **dnetstat** can obtain for both local and remote nodes includes:

1. Is DNET 'alive' at the Node?
2. The Number of active and inactive DNET Processes (long and short formats; Streaming and/or Connectionless Options)
3. Statistics of DNET Use at the Node
4. DNET Routing Tables at the Node

The general form of the **dnetstat** command is as follows:

dnetstat [dnet_network] [dnet_host] [options]

If the network and host arguments are both omitted, the local host is assumed by default.

If the status of a host on the local DNET network is required, only the **dnet_host** argument is required (local network is understood).

10.1 Testing if DNET is alive

As an introduction to dnetstat, try using the 'ping' option on your local host. This is done by typing

dnetstat -p

If DNET is 'running' on the local machine, the following message will appear:

DNET is ALIVE at dnet_network dnet_host***

This response indicates that

1. At least one DNET PVC Master Server is running on the local node
2. The DNET Datagram Master Server is running on the local node

If DNET is not running normally on your system, the following message will appear

Timed out waiting for response

Now try using dnetstat to 'ping' another DNET host on the local or a distant DNET network.

If this is successful, you are further assured not only is the DNET software running at that host, but also that the DNET datagram service is operating (at least between your machine and the distant host).

10.2 Obtaining Status of DNET Servers

dnetstat may be used to obtain the status of DNET processes at local and remote DNET nodes.

This information may be obtained in the following formats

1. Connection Oriented Services only
2. Connectionless (Datagram) Services only

3. Both Services
4. Short Display Format - types, number avail, and state of servers
5. Long Format - short format info + (Process IDs) and Start/Idle Times

The default format is

Both Services
Short Display

The short listing of server status is shown below. The command used is:

dnetstat [network] [host]

********* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:**

Srv Type	Image	PS	Av	Max	S#
dmstcp					
dechod	dechod	1	1	1	1
drexecd	drexecd	1	1	1	1
dtftpd	dtftpd	1	1	1	1
dncld	dncld	3	3	3	3
dlogind	dlogind	1	1	1	1

********* DNET CONNECTIONLESS (Datagram) STATUS at: dnett1 sun3:**

ProcName	S	StartTime
dgstcp	1	Aug 1 10:44
	1	Aug 1 10:44
dnstatd	1	Aug 1 10:44
dnetstat	1	Aug 1 10:46

A longer listing of the server status may be obtained using the l (long) and c (connection) options.

dnetstat [network] [host] -lcd

********* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:**

Srv Type	Image	PS	Av	Max	S#	PID	IU	St Time	Idle Since
dmstcp						5489		Aug 1 10:44	
dechod	dechod	1	1	1	1	5491	N		Aug 1 10:44
drexecd	drexecd	1	1	1	1	5492	N		Aug 1 10:44
dtftpd	dtftpd	1	1	1	1	5493	N		Aug 1 10:44
dncld	dncld	3	3	3	3	5494	N		Aug 1 10:44
						5497	N		Aug 1 10:44
						5498	N		Aug 1 10:44
dlogind	dlogind	1	1	1	1	5499	N		Aug 1 10:44

A long listing of the both virtual circuit and datagram server status may be obtained using the l (long),

c (connection), and d (datagram) options.

dnetstat [network] [host] -lcd

********* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:**

Srv Type	Image	PS	Av	Max	S#	PID	IU	St Time	Idle Since
dmstcp						5489		Aug 1 10:44	
dechod	dechod	1	1	1	1	5491	N		Aug 1 10:44
drexecd	drexecd	1	1	1	1	5492	N		Aug 1 10:44
dtftpd	dtftpd	1	1	1	1	5493	N		Aug 1 10:44
dncld	dncld	3	3	3	3	5494	N		Aug 1 10:44
						5497	N		Aug 1 10:44
						5498	N		Aug 1 10:44
dlogind	dlogind	1	1	1	1	5499	N		Aug 1 10:44

********* DNET CONNECTIONLESS (Datagram) STATUS at: dnett1 sun3:**

ProcName	S	PID	IPC-Name	IPCID	SIG	MSzStartTime
dgstcp	1	5482	DN_5482	1	0	0Aug 1 10:44
	1	5481	DN_5481	2	0	0Aug 1 10:44
dnstatd	1	5495	DN_5495	3	0	0Aug 1 10:44
dnetstat	1	5504	DN_5504	4	0	0Aug 1 10:45

To obtain the routing table at a particular host, enter the following command:

dnetstat [network] [host] -r

An example of output resulting from this command is:

********* DNET ROUTING TABLE at: dnett1 sun3:**

DestNet	Nxt Host	Nxt Proc	DG Protocol
dnett1	NULL	NULL	tcp
spanet	dacvax	drelaytd	tcp
starnet	dacvax	drelaytd	tcp

11. Presentation Services

DNET provides a limited presentation layer facility.

Within the DAVID environment, the single most important coding problem across heterogeneous machines is the internal representation of data. Information moved from one machine to another may only be viewed consistently if data types are faithfully "mapped" between machines.

Thus, if the transmitting machine views integers as 32 bit quantities and represents floating point numbers with 64 bits while the receiver represents these two data types as 64 and 48 bit quantities, respectively, serious misalignment of data files will occur.

The Presentation Layer Service provided by DNET is a subset of the SUN (XDR) External Data Representation and/or Existing Data conversion facilities of DAVID.

11.1 XDR Services

XDR Services are currently not available at the user level in DNET. For further information on use of XDR at the programming level, the reader is referred to the DNET Programmer's Reference Manual.

12. Glossary

The following terms are used in the description of DNET:

Applications Servers-

Servers such as File Transfer, Remote Login, Remote Execution, etc. that perform services for clients. Applications Servers are invoked on demand by clients after using the Service Assignment to obtain the name of an available server.

Connection Lock Table-

List of open connections held by process for use by its Basic Datagram I/O package. Locked connections result from user requests for Permanent Virtual Circuits.

Datagram Master Server (DGMS)-

A server process, located at each DNET host and gateway, which provides an interface to DNET clients and servers and the DNET Connectionless Datagram and Signalling Service

Datagram Protocol Servers (DPS)-

Protocol specific servers located at each DNET host and gateway, which provides an DNET Connectionless an interface to the underlying network Datagram service.

Master Server Init Table-

These tables, **tbls.msinittcp** and **tbls.msinitdec** contain initialization information for the DNET Master Servers including the type of server to be activated, the maximum # allowed at this host, and the number to make available initially, and an indication of whether the server must be prespawned. The tables are updated by the local System Administrator at the specific DNET host.

Master Server Table-

One for each DNET host, it contains information on the types and numbers of each class of DNET server actively supported on this node at any instant. Each generic server entry points to a **Server Instance Table** which lists the current specific instances of a particular class of server. It is updated by the Master Server and by specific DNET application servers.

Master Server Process (DMS)-

Processes, one per Network, managing the Master Server Table, handling server registration, server assignment, and server control. They are spawned by network start-up command files.

DNET Basic I/O package-

Included as library within an application program, it provides network i/o interface

including datagram formatting.

Gateway-

A DNET node at which communicaton protocol boundary is passed. DNET relay servers move data from one network to another performing an effective protocol conversion for streaming services. These servers are created, allocated, and used like any other DNET streaming applications servers. The Datagram Master Server, in conjunction with protocol specific datagram servers performs a similar function for DNET datagrams.

Network Command Line Interpreter-

DNET Client process that directs the execution of network commands using datagrams sent to various hosts and several servers.

myname - hostname table-

A table, **tbls.myname**, maintained in the **dnet_home** directory on each DNET node lists the DNET networks to which that host is connected and the name(s) by which the local host is known on those networks.

Network Command Language Processor-

Server that directs the execution of network commands using datagrams sent to various hosts and several servers. It is an application server, copies can be pre-spawned or spawned on demand.

Network Command Server-

Spawned by request from Command Language Processor, this Server is directed by Command Language Processor. It spawns processes and directs i/o to execute network commands.

Network Status Server-

Resides in each network host. Receives Host Status Tables, Host Alias Table, Well Known Server Tables, Connectivity Tables, and periodically sends "I am alive" messages to the Administrative host. To ensure these periodic messages are sent the Basic datagram I/O package uses a timer/wake-up signal to initiate the transmission of the message to the Network Status Client. Because this is done by the I/O package and there is a copy of this package in every process that uses network I/O the network status data is collected on a per process not per host basis.

PVC Relay

A DNET relay used in the completion of DNET Permanent Virtual Circuits (PVCs).

Relay

Special DNET application processes located in a DNET gateway which perform protocol conversion for DNET streaming service between dissimilar networks. The appropriate Master Server process 'listens' on a particular protocol boundary when idle and assigns a relay when a request for a protocol h'hop' is received from DNET..

The relays are named according to the protocol boundary which they are intended to bridge. Thus a T-D relay services requests which arrive on a TCP/IP network, relaying data to a DECnet net. Relays operate in a full duplex mode while actually in use.

Router

DNET employs a hierarchical routing strategy. Each DNET node has, for every (DNET) network known to it, information on the next DNET host to contact in order to move data toward the destination. The DNET router function uses the information in the routing table as follows: Given a destination network, host, and process, returns the next 'best' hop (network, host, process) to 'move' toward the destination.

Routing Table-

A hierarchical routing table that contains the next 'hop' from the local DNET host/network in the direction of all other DNET networks. A minimal version of this table is provided with the distribution copy of DNET. The table is currently maintained manually by the local system administrator. In the future, this table will be dynamically configured and maintained by the local DNET Network Status Server after intial startup has taken place. The routing table is named **tbls.net** and is located in the **dnet_home** directory.

Server Assignment Function-

Returns the name of an available server to a requesting Router, and updates the Domain Server Table.

Server Instance Table(s-

Lists the current specific instances of a particular class of DNET Application Server. Entries are made by the Master Server and cleared via dn_done() calls from the servers as they complete their tasks.

Server Registration Function-

This function is part of the Domain Server Process. It updates the Domain Server table with information from Servers (e.g."now in use").

DNET

USER' S REFERENCE

Version: 1.16
Print Date: 08/10/89 12:28:00
Module Name: user.ref

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

NAME

decho - dnet 'echo' client

SYNOPSIS

decho *dnet_network dnet_host*

DESCRIPTION

The decho command performs a simple demonstration and test of the DNET network. A DNET permanent virtual (streaming) connection is opened to the destination network:host. Command line input at the local host is then echoed back from the destination after each carriage return.

The command provides a convenient means of demonstrating the setup time and end-to-end performance of the DNET streaming service.

The ability to run decho depends on its presence in the Master Server Init Table for the destination host.

Command line arguments

dnet_network	name of the destination DNET network
dnet_host	name of the destination DNET host

SEE ALSO

dechod(1)

DIAGNOSTICS

The message "**Ready**" will appear after decho has succesfully connected to the specified remote node. An error message will appear if a connection can not be established.

NAME

dlogin - dnet 'remote login' client

SYNOPSIS

dlogin dnet_network dnet_host

DESCRIPTION

The dlogin command provides a remote login function over the DNET network. A DNET permanent virtual (streaming) connection is opened to the destination network:host. A standard DNET command line prompt for remote login is then presented:

DSH>

The user may then issue commands which will be understood in the 'native' environment of the destination machine.

xxxxxx

The dlogin command provides a convenient means of executing simple command line instructions on a remote machine.

The ability to run dlogin depends on its presence in the Master Server Init Table for the destination host.

Command line arguments

dnet_network name of the destination DNET network

dnet_host name of the destination DNET host

SEE ALSO

dms, dlogind(1)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

dmail - dnet 'mail' client

SYNOPSIS

dmail dnet_network dnet_host dnet_user

DESCRIPTION

The dmail command performs a simple mail transfer to another DNET user.

The ability to run dmail depends on its presence in the Master Server Init Table for the destination host.

Command line arguments

dnet_network name of the destination DNET network

dnet_host name of the destination DNET host

dnet_user user at the destination DNET host

SEE ALSO

dms, dmaild(1)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

dncl - dnet 'network command language' client

SYNOPSIS

dncl

DESCRIPTION

The **dncl** command invokes the interactive dnet network command language program. This program allows for processing of a single data stream in a distributed environment. To do this, the processing of the data stream is broken into sub command lines **SCL** (which together make up the dncl command line **CL**). The dncl CL may be entered after the dncl prompt:

dncl >

The following is a synopsis of the dncl command line:

SCL > SCL [> SCL] ...

You will note that a minimum of two SCL components are required in a CL. The reason for this will be explained when we look at the three categories of SCL components. Also note that the > symbol is used to delimit the SCL components.

The following is a synopsis of the SCL component:

[[netname::]hostname:] [*]command/file

Notice that **netname** and **hostname** are optional, although if a network name is supplied, then a host name must also be supplied. In the case where both **netname** and **hostname** are specified, a double colon must delimit the **netname** and the **hostname**, and a single colon must delimit the **hostname** and the **command/file**. Further, if the **command/file** value contains a colon, then the **hostname** must be supplied at a minimum so that the colon within the **command/file** will be ignored by dncl.

If the requested node is the current machine (the **netname** and **hostname** combination represent the current machine), and no colons appear within the **command/file** value, then **netname** and **hostname** may be omitted. Similarly, if the **hostname** machine is on the current network, then **netname** may be omitted. On dnet gateway machines remember that only one network is considered to be current. This means that although the network may be directly connected to the current machine, it can not be considered a current network.

The **command/file** portion of the SCL represents either a file or a command to be accessed on the given machine and falls into one of three categories:

- First SCL component -- must be a file
- Middle SCL component -- must be a command (precede with *)
- Last SCL component -- must be a file

As you will remember from the CL synopsis above, and minimum of two SCL components must be specified (a First SCL component and a Last SCL component). This represents the simplest form of a dncl CL and results in a file transfer without filtering. The dncl CLs of greater complexity merely represent a higher degree of filtration between the first and last SCL components. The filtration described here is provided by the middle SCL component category (a command). This command is assumed to read input from a standard location, process the input received and generate output to a standard location. Many commands can be described in this fashion (input/processing/output), but not all work with standard locations for input and output. Commands that do use standard locations and work in the input/processing/output fashion are described as being filters. To work properly as a middle SCL category SCL component, the command must also be a filter, as unpredictable results will otherwise occurr.

All middle SCL category SCL components must be preceded with an asterix (*) as shown in the SCL synopsis above.

The UNIX operating system is rich with existing filters to perform a variety of tasks. These filters are comparatively rare in the VMS operating system. Despite this, filters may be created for VMS with C language programs by using the predefined **stdin** and **stdout** streams with the standard I/O package.

SEE ALSO

dtftp(1), dsh(1)

DIAGNOSTICS

After successful completion of a dncl CL, the following message will be displayed:

ACKCOMP received.

This means that the ACKCOMP (ACKnowledge COMPletion) packet has been initiated by the last SCL category driver, and has been successfully passed back through all intermediate SCL components to be successfully received by the dncl command invoked by the user.

If the ACKCOMP received message is not displayed, then a cryptic error message will be displayed describing the reason for failure. If the error message is preceded by **dncld:**, then this means that the error occurred at a possibly remote node, and this message was propagated back to be viewed by the user.

A common form of error message is:

No route to netname::hostname:dncld

This indicates that the node specified could not be found from the current location. Two things should be remembered to help to solve this problem:

1. You may not have specifed the node name portion of the stated SCL, and the default may have been used.
2. The node is always relative to the node on the previous SCL component. The first SCL is always relative to your current node. As an example, if the first SCL was specifed as: **spanet::iaf:sys$login:myfile** , and the second SCL was: ***sort**, then it would try to spawn the sort filter on the spanet::iaf node.

CAVEATS

Never make assumptions about current location within a file system on any node when creating SCL components. Absolute pathnames or logical names must be used for files. For commands, absolute pathnames or logical names must also be used, but on UNIX operating systems, the PATH environmental variable may be set by the dnet administrator before the dncl drivers are initiated so that they can be forced to look in non-normal locations for UNIX filters.

NAME

dnetstat - obtain dnet network status

SYNOPSIS

dnetstat [*dnet_network*] [*dnet_host*] **[-acdfhlnprs]**

DESCRIPTION

The **dnetstat** command allows the display of various DNET-related data structures. Information may be displayed in various forms, depending on the option which is specified. **dnetstat** can be used to determine the status of all DNET servers, routing tables, and server usage statistics.

Information may be displayed for the local DNET node or may be retrieved and displayed for other DNET nodes.

Options:

dnet_network - the DNET network of the DNET host from which information is desired; if omitted, local network is assumed

dnet_host - the DNET network of the DNET host from which information is desired; if both network and host omitted, local host is assumed

If none of the below options is specified, the defaults **local_host** & **[-cd]** are assumed

-a Display all available information (in long format)

-c Display Status of Connection (Streaming) Servers

-d Display Status of Datagram (Connectionless) Servers

-f Display PIDs, etc. in alternate (Decmimal/Hexidecimal) format; allows optional conversion between machines with different display formats

-h Display help on options for **dnetstat**

-l Display other specified options in **long** or extended format

-n show DNET map (network, host)

-p ping the specifed host - i.e. test if DNET is alive on the specified host **p** overrides all other options. If successful, the message:

DNET is Alive at dnet_network dnet_host

is printed on the terminal If the 'ping' operation is unsuccessful, **dnetstat** will usually timeout waiting for the response from **dnstatd**.

Timed out waiting for response

-r show DNET routing tables for the specified node

-s show per-DNET server statistics (dtftp, drexec, dmail, dncl)

SEE ALSO

dnstatd, tbls.msinitdec, tbls.msinitdec, tbls.net

DIAGNOSTICS

The call fails if:

Specified host is not up.

DNET is not operating on the specified host

dnstatd is not operating on the specified host

In each of the above instances, **dnetstat**, will report:

Timed out waiting for response

NAME

drexec - dnet 'remote execution' client

SYNOPSIS

drexec *dnet_network dnet_host*

DESCRIPTION

The drexec command performs a simple remote execution function over the DNET network. A DNET permanent virtual (streaming) connection is opened to the destination network:host. A standard DNET command line prompt for remote execution is then presented:

DREXEC>

The user may then issue commands which will be understood in the 'native' environment of the destination machine.

xxxxxx

The drexec command provides a convenient means of executing simple command line instructions on a remote machine.

The ability to run drexec depends on its presence in the Master Server Init Table for the destination host.

Command line arguments

dnet_network name of the destination DNET network

dnet_host name of the destination DNET host

SEE ALSO

dncl(1)

DIAGNOSTICS

NAME

dtftp - dnet trivial file transfer client

SYNOPSIS

dtftp [*dnet_network*] **[dnet_host]**

DESCRIPTION

The dtftp command allows the transfer of files to and from remote DNET machines. Information may be displayed in various forms, depending on the option which is specified.

Information may be displayed for the local DNET node or may be retrieved and displayed for other DNET nodes.

Command line options

dnet_network	name of the destination DNET network
dnet_host	name of the destination DNET host

Commands

cd XXX	change the default directory on the remote host to XXXX
get	retrieve a file from the remote to the local host
help	display help message for available **dtftp** commands
lcd XXX	change the default directory on the local host to XXX
lpwd	list the current directory on the local host
ls	list the contents of the current directory on the remote host
lls	list the contents of the current directory on the local host
mode	Allows specification of **binary** or **ASCII** mode
put	transmit a file from the local to the remote host
pwd	list the current directory on the remote host
quit	end the file transfer session

SEE ALSO

dtftpd(1)

DIAGNOSTICS

DNET

PROGRAMMER' S GUIDE

Version: 1.26
Print Date: 09/05/89 11:21:16
Module Name: prog.gui

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

CONTENTS

1. Introduction 1
   1.1 DNET Modes of Operation 2
2. DNET Host Software 3
   2.1 Overview of a DNET Host 3
      2.1.1 Basic I/O Function Library 6
   2.2 Client/Server Relationships in DNET 6
      2.2.1 Definitions 6
      2.2.2 Types of Clients 7
      2.2.3 Types of Servers 7
      2.2.4 Control of Servers 7
      2.2.5 Number and Types of Servers 7
3. DNET Private Virtual Circuits Operation 10
      3.0.1 Client/Server Conversation 13
      3.0.2 Closing a Client/Server Conversation 14
4. Writing Connection Mode Services 15
   4.1 DNET Client Design Issues 15
      4.1.1 General Rules for Coding DNET PVC Client 15
      4.1.2 Detailed Discussion 16
   4.2 DNET Server Design Issues 18
      4.2.1 General Rules for Coding DNET PVC Server 18
      4.2.2 Detailed Discussion 19
   4.3 Connectionless Datagram Service in a Streaming Application (if required) 19
   4.4 An Example Streaming Application 20
5. Connectionless Mode Services 25
   5.1 Introduction 25
      5.1.1 Connectionless Datagram Formats 26
   5.2 Details of Datagram Services 27
      5.2.1 Registration with the local Datagram Master Server 27
      5.2.2 Sending a Datagram 27
      5.2.3 Receiving Datagrams 28
   5.3 Return Receipt Service 29
   5.4 Signalling Services 29
      5.4.1 Sending a Signal 30
      5.4.2 Receiving a Signal 30
   5.5 A Connectionless Service Example 31
      5.5.1 Datagram Protocol Servers 35
   5.6 Signalling 36
      5.6.1 Sending Signals 36
      5.6.2 Delivery of Signals 36
6. DNET Error Handling 37
7. Routing 38
8. Interprocess Communication 39
9. Presentation Layer Services 40
   9.1 XDR 40
      9.1.1 Issues in the Use of XDR 40
      9.1.2 The XDR Handle (Control Structure) 41
      9.1.3 Creation of the I/O Datastream 41
      9.1.4 Encoding/Decoding of Data using XDR library 42

9.1.5 Example - use of XDR in dnetstat 45
9.2 Transferring arbitrary files using XDR 51
9.3 Existing DAVID Presentation Service 51
9.3.1 Virtual Data Format for DNET Transmission 52
10. Standard DNET Code Organization 53
10.1 Standard Directory Structure 53
10.2 Variation for VMS Installations 53
11. Compiling & Making DNET Applications Programs 54
11.1 General Strategy 54
11.2 Setting DNET Compile Time Environment Variables 54
11.3 Making UNIX Version 54
11.3.1 BSD Systems 54
11.3.2 Example Make File 54
11.4 Making VMS Version 58
11.4.1 General 58
11.4.2 MicroVAX II 58
11.4.3 NASA-GSFC VAXes (IAF, DFTNIC, etc. using Excelan TCP) 58
11.5 Making individual files 58
12. Debugging DNET applications 59
13. DNET Error Codes 60
14. Glossary 62

1. Introduction

This Guide is intended to provide the information necessary for a programmer to write DNET applications using the DNET Basic I/O package.

The generation of 'ordinary' applications should be handled by the information in this guide. Programming details for special DNET applications and internal functions, such as Master Servers, Protocol Specific Datagram Servers, and Network Command Language Internals are discussed in the Technical Guide and Reference.

A discussion of DNET operation relevant to writing standard applications is presented first, followed by general issues of interest in writing clients and servers. These discussions are followed by basic 'code' templates and specific application examples which illustrate both streaming and connectionless DNET applications. Also included are procedures for 'making' DNET code on the various target machines.

1.1 DNET Modes of Operation

The basic function of DNET is to provide a reliable communication interface between any application pair (client/server) running over DNET. Depending upon the particular usage, this service may be viewed as employing either a client/server or a task/task model of operation. The task to task model assumes two (or more) arbitrary processes (tasks) on separate DNET hosts may communicate with one another providing they follow DNET conventions and use the DNET basic I/O package or Network Command Language in order to communicate. The service is offered in both connection-based and connectionless modes.

The connection based mode provides a dedicated channel with private gateways and relay processes. It provides high quality service, but requires that a set of processes be spawned and connections be established specifically for this communication session. The connection based DNET service establishes permanent virtual circuits between communicating tasks for the duration of the communication session. To use the connection based service, the function, "dn_open()", is required. It creates a circuit that supports DNET data streaming mode which is useful in applications such as remote login, where rapid, interactive processing is required. Once such a streaming connection has been established, DNET becomes a "smart wire" between the communicating tasks; i.e. user program data moves over the open connection as if it were simply a hardwire link. The calls "dn_write()" and "dn_read()" may then be used to exchange data over the network.

DNET also provides a reliable, connectionless, datagram service. In connectionless mode, processes may communicate with one another without any (apparent) need for a circuit connection to be established. Data are transmitted in units (datagrams) comprising data prefixed with headers containing source/destination information to be used by the communications software during the transmission of the data.

2. DNET Host Software

This section describes the software components provided at DNET Hosts. There are four major topics:

1. Overview of a DNET Host
2. DNET Basic I/O Library Functions
3. Supporting Software for the DNET Host
4. DNET Applications

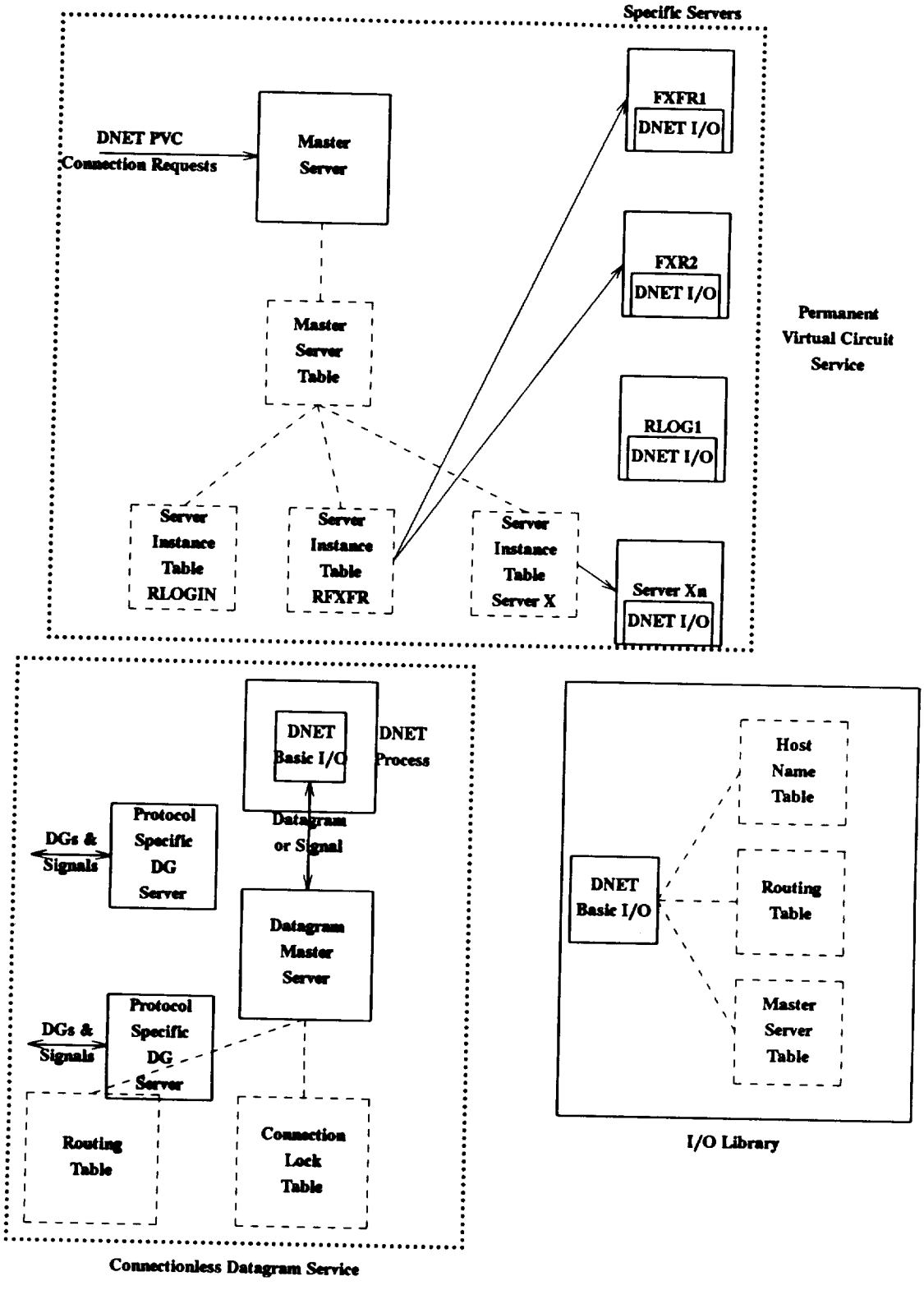
More advanced software associated with DNET internals is discussed in the **DNET Technical Guide.**

2.1 Overview of a DNET Host

Any computer connected to one of the networks served by DNET may become a node on DNET provided the following conditions apply. A DNET host machine must:

1. be resident on a specific existing network (e.g. TCP/IP Net X, SPANET, etc.) which DNET considers as one of its 'domains'
2. have DNET Host Software Installed & Operational
3. be "activated" by the local System Administrator (see DNET ADMINISTRATOR'S GUIDE)

The elements in the DNET host are shown in the following diagram:



Major DNET Host Elements

A brief description of these elements follows:

Software Components

1. **DNET Basic I/O Package** - this is a library of function calls which provides basic capability to generate, route, read, and write DNET datagrams and signals and to establish and drop DNET permanent virtual circuits.
2. **DNET Master Server** - This process controls the spawning and allocation of all DNET application servers (see below). It is a Well Known DNET Server
3. **DNET Application Clients (as may apply at particular host)** -
   - **DNET Network Command Interpreter (NCI)** - A special command line interpreter which allows DNET network commands to be invoked from the local machine.
   - File Transfer
   - Remote Login
   - Mail
   - Network Status
4. **DNET Application Servers (as may apply at particular host)** -
   - DNET Network Command Server (NCS) - allows interpretation of DNET Network Commands "distributed" from a DNET Network Command Interpreter
   - File Transfer
   - Remote Login
   - Mail
5. **DNET Datagram Server** - This server provides an interface between clients and servers and the DNET connectionless datagram service
6. **per Protocol Datagram Server** - These servers provide interfaces to the underlying networks for Datagram Service; the interface method is picked to be the most efficient available in a particular circumstance
7. **DNET Network Status Server** - This server uses the datagram service to provide status information in response to requests from the **dnetstat** network status client.

Tables and Variables

1. **Master Server Init Table** (tbls.msinittcp & tbls.msinitdec) - This is a file containing the initialization information for the Master Server. It is loaded into the Master Server Table when the DNET software is started on the local node.
2. **Master Server Table** - information on the server programs controlled by a Master Server, including status and connecting links to Master Server. - This table is generated in memory by the Master Server Process based on contents of the Master Server Init Table.
3. **DNET Server Instance Table(s)** - These tables list detailed instances of Specific DNET servers under control of the Master Server at this DNET host. There is a separate SIT for each type of server available at this node. These tables are internal to the Master Server but may be viewed using the **dnetstat** process (see USER's and ADMINISTRATOR's guides).
4. **Routing Table(s)** (tbls.net) - table identifying the gateways from this host to every other network or to networks that lead to all the other networks.
5. **Connection Lock Table** - table containing the channel numbers of permanent virtual circuits used by the I/O Package and their correspondence to user program logical streams

6. **Hostname Table** (tbls.myname) - File containing name of local node and its DNET network
7. Services Files
   1. UNIX

      The standard file

      /etc/services

      must contain the following entries:

      ```
      5279    dms/tcp          # DNET PVC Master Server
      5279    dgsudp/udp # DNET UDP Datagram Server
      ```

   2. DEC

      For VMS Systems, the DNET master servers are automatically 'registered' as network objects when DNET is started.

2.1.1 Basic I/O Function Library

The function calls provided in the DNET basic I/O library are summarized in the following table:

Generic Operation	VIRTUAL CKT Client	VIRTUAL CKT Server	Datagram	SIGNAL
Estab. Connect.	dn_open	dn_getclient		
Write	dn_write		dn_cwrite	dn_signal
Synch Read	dn_read		dn_cread	Dest Oper Sys
ASynch Read			dn_cdg_handler	Dest Oper Sys
End Connect.	dn_close	dn_done,dn_close	dn_cdone	

These functions are described in the following sections according to the type of service (PVC, Connectionless Datagram, or Signal) which they support.

2.2 Client/Server Relationships in DNET

Most applications which use DNET interact via conversations between a client process and a server process. This section describes the general strategy by which such client-server relationships are established and operate within DNET.

2.2.1 Definitions

- **Client** - initiating process; DNET communications initiated by the client or processes which it starts; requests service from a distant server process;

- **Server** - local or remote process which provides the service desired by the client process; the server must be spawned prior to responding to a request for service from a client process;
- **Master Server** - a process located at each DNET host which services all requests from clients for DNET Application Servers which use PVC service, including DNET PVC Relays at Gateways. Each Master Server listens on a well-known port on a specific underlying network. Hence, DNET gateway machines will contain master servers for each protocol actively used by DNET.
- **DNET Datagram Master Server (DGMS)** - an independent master Server located at all DNET host machines which allows the tranmission, reception, and/or relay of DNET connectionless datagrams and signals. Processes which wish to use the DNET connectionless service must pre-register with the DGMS. **Protocol Specific Datagram Servers** - These servers provide protocol specific interfaces to existing networks for the DNET connectionless service.

2.2.2 *Types of Clients*

Four major client "types" are expected in DNET:

1. DNET Applications
2. DNET Network Command Interpreter
3. User Defined Clients (new DNET applications)
4. Other existing Clients (possible future expansion); e.g. telnet, FTP, etc.

2.2.3 *Types of Servers*

There are two application server types defined within DNET:

1. **DNET Application Servers** - called by client processes, these service providers include a DNET Basic I/O package. For all these services (File Transfer, Network Command Server, other application servers) there is a process which spawns copies of them and assigns the copies to clients on request. This controlling process is the "DNET Master Server".
2. **Other Servers (user defined, etc.)** - spawned via DNET network command server (net_com_serv) these servers do not contain the DNET Basic I/O Package. They depend on the network command server to interface with DNET.

2.2.4 *Control of Servers*

DNET servers which require streaming service are under the control of the **DNET Master Server(s)** at each DNET host. These servers may be either prespawned or spawned on demand depending on the type of host and local system considerations.

Bidirectional connectionless service is also available to these servers if they register with the Datagram Master Server. Details of connectionless operations are provided in a later section.

2.2.5 *Number and Types of Servers*

The system administrator on a particular DNET host controls the number and types of DNET servers which operate on that host.

The number and types of servers are determined by the DNET Master Server Table Init file:

This is a 'flat' ASCII file. Entries in the file appear on separate rows and have the format as follows:

DNET Master Server Init Table				
Server Type	Image Name	# Prespawned	Max #	Init #
dechod	dechod	1	8	3
dtflpd	dtflpd	1	9	4
drexec	drexec	1	1	1
dnstatd	dnstatd	1	1	1
dncld	dncld1	1	10	5
dlogind	dlogind	1	10	5
dmaild	dmaild	1	10	1

The number of prespawned servers is specified in column 3.

The Maximum (permissible) number of servers of this type is specified in column 4

Column 5 contains the number of servers to be started when DNET is first started

Servers may be added or deleted by editing this file (DNET admin privileges required)

Further discussion of the significance of these entries is provided in the following sections.

A separate Master Server Init File is required for each protocol connection at a DNET host. Thus, at a VAX which is connected to both a TCP/IP and a DECnet Network, there must be two such tables **tbls.msinittcp** and **tbls.msinitdec.**

2.2.5.1 Prespawning of Servers

In order to improve the efficiency of response for DNET service requests on VAX machines, certain DNET servers may be 'prespawned' prior to service requests.

The number and type of prespawned servers is specified in the Master Server Init Table File described in the preceding section.

Possible algorithms for spawning and assignment are:

1. At network start up, spawn a number of copies of the servers, according to the contents of the **DNET Master Server Init Table** keeping their process id's for later use in forming the process names to give to clients. After allocating a server to a client, spawn another to replace it.
2. For less frequently used services- Spawn only when a client requests a server. This is the Transient Server.
3. For very frequently used services- Spawn the maximum number desired and have servers listen for the next client when they complete their service for a client, and at the same time notify the Master that they are ready for assignment.

2.2.5.2 *Maximum Number of Servers*

This parameter controls the maximum number of simultaneous copies of a particular server which are allowed at the local host. This number can be adjusted by the system administrator according to conditions on the local system.

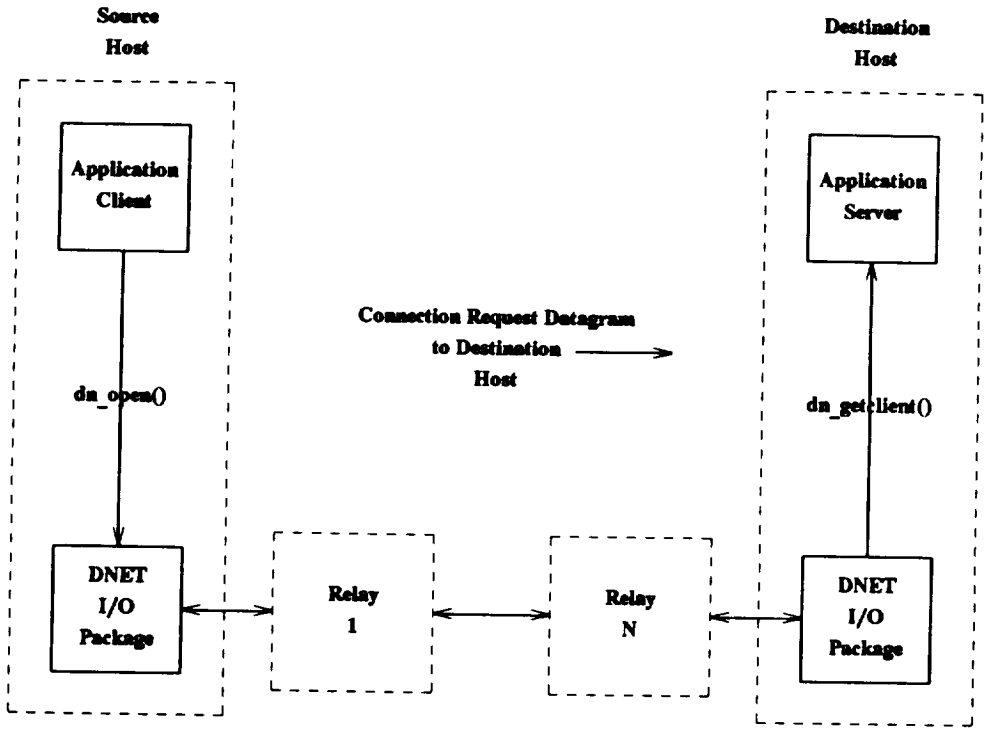
3. DNET Private Virtual Circuits Operation

To create "private virtual (streaming) circuits" DNET establishes permanent connections on all hops from the client to the server process. In establishing the connections necessary for these private circuits, private relay processes and private communications connections are used. The establishment of the required connections and gateway relays can all be done using the standard DNET open (dn_open) function which sets up a chain of **DNET PVC Relay** processes. dn_open() operates by forwarding a connection request datagram through DNET from client to requested server. Intermediate PVC relays read the connection request datagram and open a connection to the next host/process using the routing information in the datagram. The private relay/gateway processes described above are provided as special DNET application servers and are included in the routing information in the header of the connection request datagram. In this way the private virtual circuit will be created by the Basic I/O package as it transmits the datagram and opens connections to successive host/processes. (Opening a connection to a process will cause the process to be spawned if it is not already extant.)

Once a private virtual DNET circuit has been established, data is transmitted and received using the functions dn_write() and dn_read();

To close the connections created for the private virtual circuits described above, the process that originated the connections simply calls the function **dn_close()** which causes the PVC to be dropped.

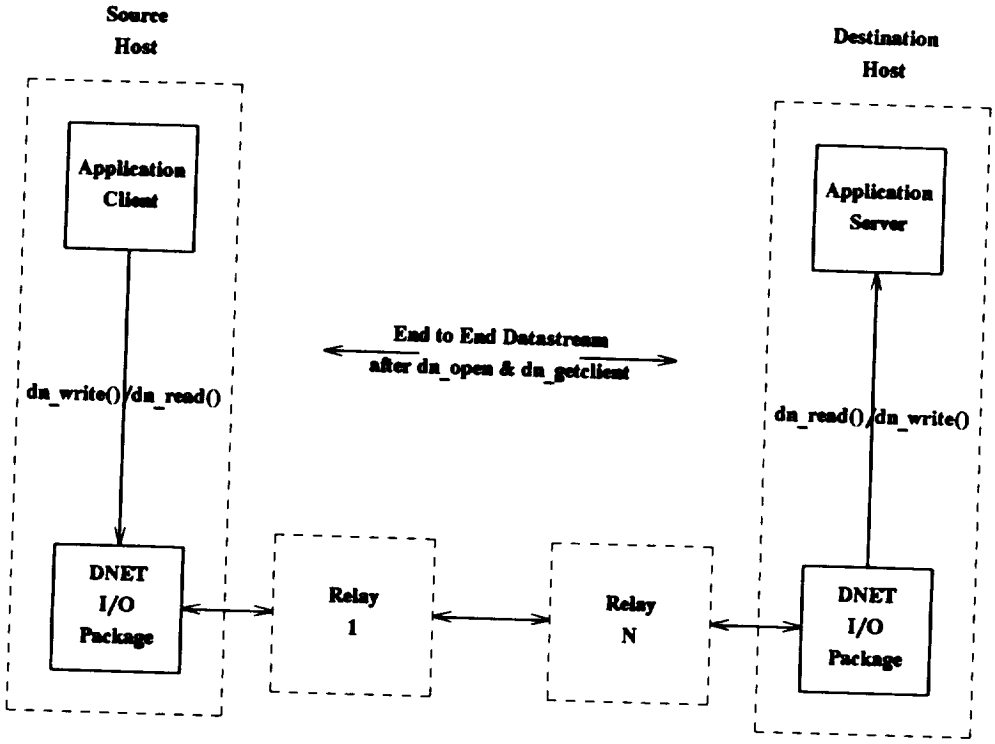
An overview of the setup for a DNET PVC is shown in the following diagrams: A connection request is first made by the client by calling dn_open():



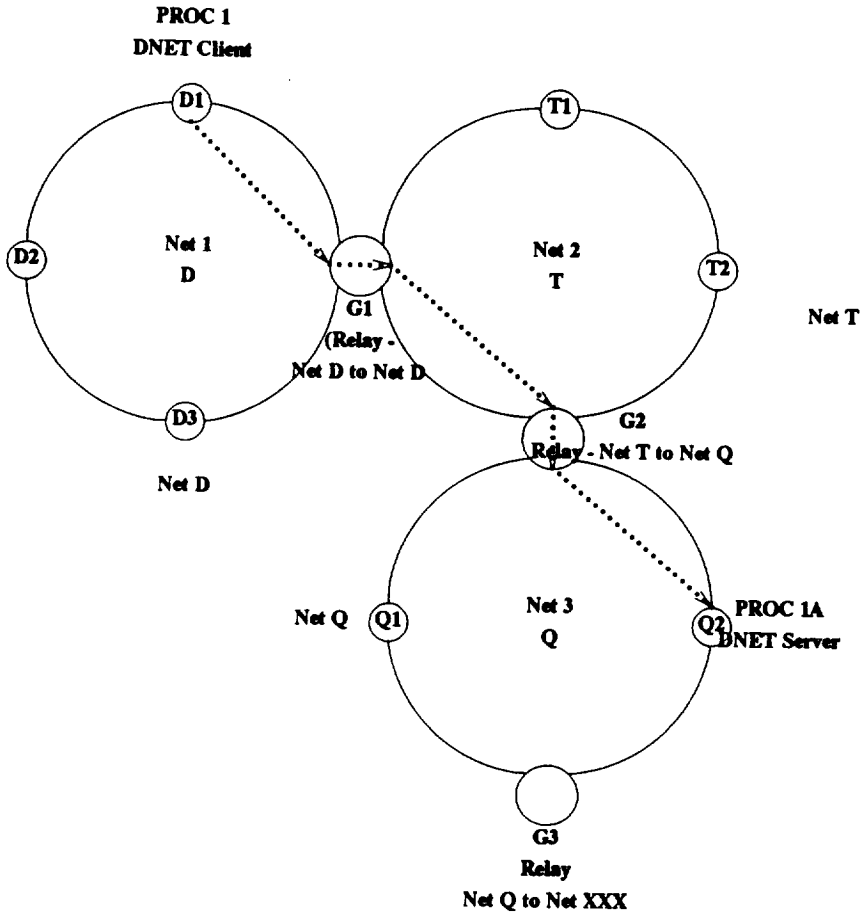
The format of the connection request datagram is shown below:

Field	#
0	Type = Connection Request
1	origin_net
2	origin_host
3	origin_service (process)
4	dest_net
5	dest_host
6	dest_service (process)
7	next_service
8	callback_flag
9	callback_port
10	callforward_stream (either channel or file descriptor)

dn_open() returns a data stream to the client only when a virtual connection has been established with the desired server. Once **dn_open()** returns successfully, the client and server may each use the functions dn_write() and dn_read() to read and write data streams to one another as shown in the following diagram:



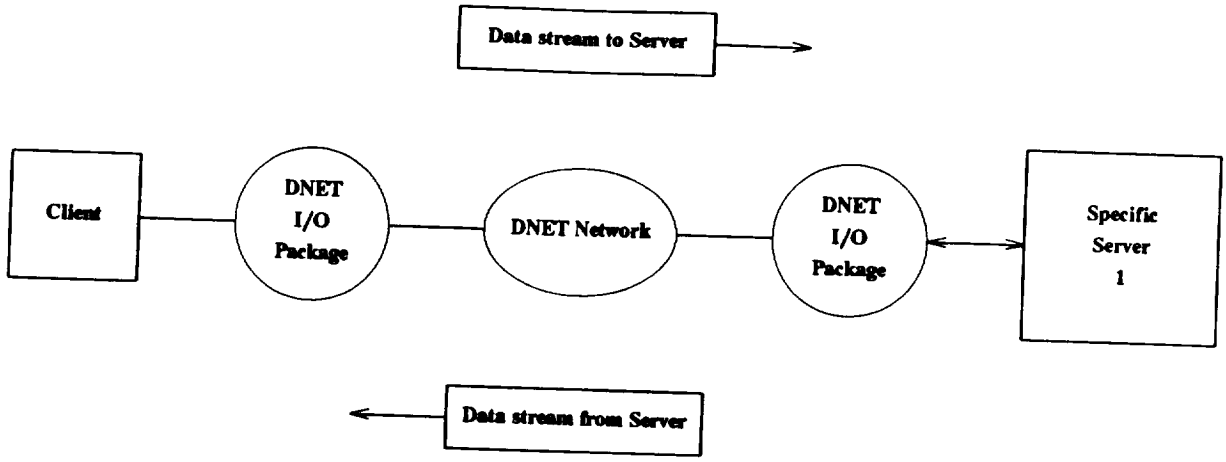
The following diagram shows schematically what a private virtual circuit would look like in the heterogeneous network example:



In this example, two DNET PVC Relay processes are employed, at gateways G1 and G2 in order to complete the virtual circuit.

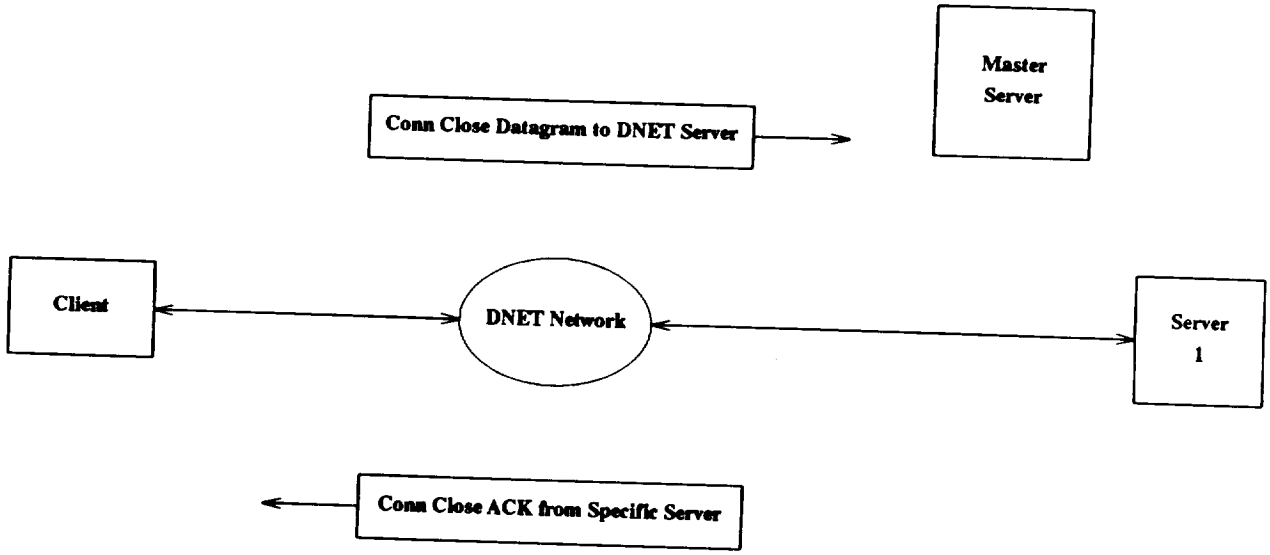
3.0.1 Client/Server Conversation

Once the PVC is 'open' data is streamed between client and server processes:



3.0.2 Closing a Client/Server Conversation

At the conclusion of a session, the DNET permanent virtual circuit may be closed by calling **dn_close()**. Either the client or server process may call **dn_close()**.



4. Writing Connection Mode Services

4.1 DNET Client Design Issues

DNET Clients are executable processes located in the directory **dnet_home/bin**. This directory should be included in the user's path for UNIX systems. On VMS systems, 'paths' to each client are defined in the **dnlogin.XX** file which should be executed by the login.com file as part of the user's login sequence.

The following defines the procedure for writing ordinary application client processes.

4.1.1 General Rules for Coding DNET PVC Client

The general form for the coding of a DNET Application Client is as follows:

```
#define SERVICE_NAME "dhackd"

main ()
{
        User-Defined Initialization

        processname = "XXXXX"  /* this is the mnemonic for the process */

        dn_init()
        .
        .

        chan = dn_open()
        .
        .
        .

        Application Code

        dn_write(chan,...)
        dn_read(chan,....)
        .
        .
        .

        dn_close()

        return/exit/bottom of client loop
}
```

The several elements in this 'standard' form are discussed in the following sections:

4.1.2 Detailed Discussion

4.1.2.1 SERVICE_NAME and progname

SERVICE_NAME is used to indicate the name of the server to which this client will connect. SERVICE_NAME is typically passed in the process field of dn_open.

progname defines the name of the client. This variable is used internally by the client and is used to name an optional 'log' file.

4.1.2.2 User Initialization

This is any application specific initialization at the option of the programmer.

4.1.2.3 Initialization - dn_init()

This is a mandatory function call which sets up the necessary run-time tables for the application. The global variable *progname* must be called.

4.1.2.4 Defining a Virtual Circuit - dn_open()

dn_open() is used by client processes to request a Private Virtual Circuit connection to the specified service a given network and host. The function does not return until a path to the destination has been opened or an error occurs (channel cannot be opened, timeout, etc).

```
dn_open
chan = dn_open(net, host, service)
        int chan;       /* A channel number;
                           used in subsequent read and write calls */
        char *net;      /* A DNET network name */
        char *host;     /* A DNET host name */
        char *service;  /* A  DNET service */
```

Once dn_open() returns a channel, the PVC is assumed to be established. The 'open' chan may be used as a 'file descriptor' in DNET read and write operations.

4.1.2.5 Using a Defined Virtual Circuit

4.1.2.5.1 Blocking vs Non-Blocking Operations

DNET read and write operations may be either blocking or non-blocking. Blocking operations are those which do not return until a certain event has occurred. Thus, a blocking 'read' or 'write' is one which waits until a specified number of bytes have been read or written, respectively.

In non-blocking operations, the action may be initiated without waiting to determine if it completes. A non-blocking operation might also be a 'poll' which simply checks for example, whether data has arrived and needs to be read.

4.1.2.5.2 Reading and Writing on a Virtual Circuit

DNET permanent virtual circuits service provides functions which closely approximate the UNIX system calls read() and write().

```
dn_write
nbytes = dn_write(chan,buf,nbytes)

        int nbytes;     /* The number of bytes, including DNET headers,
                           that was written on the given stream. */
        int chan;       /* I/O channel returned from dn_open  */
        char *buf;      /* The data that is to be sent. This function
                           prepends the data with a DNET header. */
```

```
dn_read

Synchronous (Blocking) read

nbytes = dn_read(chan, buf, count)
        int nbytes;     /* The number of bytes, including DNET headers,
                           that was read from the given stream. */
        int chan;       /* A pointer to an I/O structure that was
                           previously opened by dn_open() */
        char *buf;      /* A result parameter where the datagram, in
                           string format, is placed; this buffer
                           contains the DNET headers. */
        int count;      /* The maximum number of bytes to receive. */
```

4.1.2.6 Closing a Virtual Circuit - dn_close()

The function dn_close() closes a DNET PVC communications channel; it can be used in both clients and servers.

```
dn_close
status = dn_close(stream)
        int status;     /* An indication of success or failure */
        int chan;       /* A channel structure that was
                           previously opened using dn_open() */
```

4.2 DNET Server Design Issues

In network communications applications, the server the server can be either be permanent or transient. The choice is transparent to the client.

A permanent server is one that is initiated at network start up time and when not otherwise occupied, listens for, accepts connections and performs services until the network is shut down.

A transient server is one that listens for connection requests using one process and when a request is received, starts a process that is dedicated to the client. The listening process can either be one which performs a particular server application or it can be one that manages many servers, listening for requests to any of them and spawning or reactivating the appropriate servers.

A communications server application can be permanent on one host (a VAX, for instance, where process start-up is slow) and transient on another (most UNIX systems, where processes start quickly). Control of pre-spawned and demand spawned processes is handled by the DNET PVC Master Server according to entries in the Master Server Init Table(s) discussed elsewhere.

4.2.1 General Rules for Coding DNET PVC Server

The following is the 'skeleton' code for a DNET PVC Server:

```
#define SERVICE_NAME "dhackd"  /* name of server associated with this client */

main()
{

User Static Initialization

progname = "dhack";              /* name of this server */

dn_init();

While ( chan = dn_getclient( ) );

        User Dynamic Initialization

        Application Code
  .
  .
  .
        dn_read(chan,....);
        dn_write(chan,....);
  .
  .
  .

dn_done(); /* notify MS that session is over;
                    loop to getclient for another request */

return/exit/bottom of server loop

dn_close(chan);
}
```

4.2.2 Detailed Discussion

4.2.2.1 SERVICE_NAME & progname

SERVICE_NAME is used to indicate the name of the server to which this client will connect. SERVICE_NAME is typically passed in the process field of dn_open.

progname defines the name of the client. This variable is used internally by the client and is used to name an optional 'log' file.

4.2.2.2 dn_init()

Same role as for DNET Clients.

4.2.2.3 dn_getclient()

The function **dn_getclient()** performs a role for DNET servers parallel to than played by **dn_open()** for DNET clients. The 'idle' server waits on dn_getclient() for notification of a service request from the Master Server.

```
dn_getclient
chan = dn_getclient(service, usrbuf, pusrbuflen)
        char* service;
        char* usrbuf;
        char* pusrbuflen;
```

4.2.2.4 dn_write() / dn_read()

Same function as for client.

4.2.2.5 dn_done()

```
dn_done
        dn_done is called by each DNET Application Server before exit
        to indicate to the local Master Server that it has
        completed its task and is available for use
```

4.2.2.6 dn_close()

Same function as for client processes.

4.3 Connectionless Datagram Service in a Streaming Application (if required)

DNET applications which are primarily streaming based may nevertheless have need to use the optional connectionless service. This service may be used by following the rules outlined in the sections on DNET connectionless service later in this guide.

4.4 An Example Streaming Application

The following is an example of a typical client/server pair (**decho** & **dechod**) written using the DNET facilities. The reader is also referred to the DNET source code listings for examples from the other applications.

The user interface to **decho** is described in the DNET User's Guide.

The echo application establishes a permanent virtual circuit with a specified (remote) host. The client then accepts input from the user a line a time. When carriage return (CR) is pressed, the input is sent over the PVC to the echo server which returns it immediateally to the client where it is displayed on the next line of the screen.

Input lines are entered indefinitely until an end of file is issued by the user (Ctrl-D for UNIX, Cntrl-Z for VMS).

```
/*****************************************

decho.c

*****************************************/
#define MAINPROGRAM

#include <stdio.h>
#include "dnet_env.h"
#include "dnet.h"
#include "dnet_errno.h"

#ifdef DN_EVMS
#include <file.h>
#endif
#ifdef DN_EUNIX
#include <fcntl.h>
#endif

#define SERVICE_NAME "dechod"

main (argc, argv)
int     argc;
char    *argv[];
{
        int     channel, fid;

        if (argc < 3) {
                fprintf(stderr, "Usage: %s net node [files]\n", argv[0]);
                return;
        }

        progname = "decho";

        if (dn_init() < 0) {
                dn_error("dn_init");
                exit(1);
        }

        fprintf(stderr, "Trying to open connection to %s %s\n", argv[1], argv[2]);

        channel = dn_open(argv[1], argv[2], SERVICE_NAME);

        if (channel<0) {
                dn_error("dn_open");
                exit(1);
        }

        fprintf(stderr, "Ready.\n");

        if (argc == 3)
                echo(channel, 0);
        else    {
                while (--argc >= 3)  {
                        if ((fid = open(argv[argc], O_RDONLY)) == -1){
                                perror(argv[argc]);
                                continue;
                        }
                        echo(channel, fid);
                        close(fid);
                }
        }
        dn_close(channel);
        exit(0);
}
```

```
echo(channel, fd)
int     channel, fd;
{
        char    buf[2*DN_BUFSIZ + 1];
        int     nread, nrcvd, nsent;

        while ((nread = read(fd, buf, sizeof(buf) - 1)) > 0){
                buf[nread] = NULL;
                if (debug)
                        fprintf(stderr, "dechox after read from stdin, nread is %d0,
                                nread);
                if ((nsent = force_write(channel, buf, strlen(buf), nread)) != nread)
                        break;
                if (debug)
                        fprintf(stderr, "dechox after dn_write, nsent is %d0, nsent);
                if ((nrcvd = force_read(channel, buf, sizeof(buf) - 1, nsent)) != nsent)
                        break;
                if (debug)
                        fprintf(stderr, "dechox after dn_read, nrcvd is %d0, nrcvd);
                if ((nsent = write(1, buf, nrcvd)) != nread) break;
        }
}

force_write(chan, buf, buflen, force)
int     chan;
char    *buf;
int     buflen, force;
{
        int     nbytes, ret;

        nbytes = 0;
        while (nbytes < force && nbytes < buflen){
                if ((ret = dn_write(chan, buf + nbytes,
                                buflen - nbytes)) < 0){
                        dn_error("dn_write");
                        return(-1);
                }
                nbytes += ret;
                if (debug)
                        fprintf(stderr, "force_write: force=%d nbytes=%d0,
                                force, nbytes);
        }
        return(nbytes);
}

force_read(chan, buf, buflen, force)
int     chan;
char    *buf;
int     buflen, force;
{
        int     nbytes, ret;

        nbytes = 0;
        while (nbytes < force && nbytes < buflen){
                ret = dn_read(chan, buf + nbytes, buflen - nbytes);
                if (ret < 0) {
                        dn_error("dn_read");
                        return(-1);
                }
                nbytes += ret;
                if (debug)
                        fprintf(stderr, "force_read: force=%d nbytes=%d0,
                                force, nbytes);
        }
        return(nbytes);
}
```

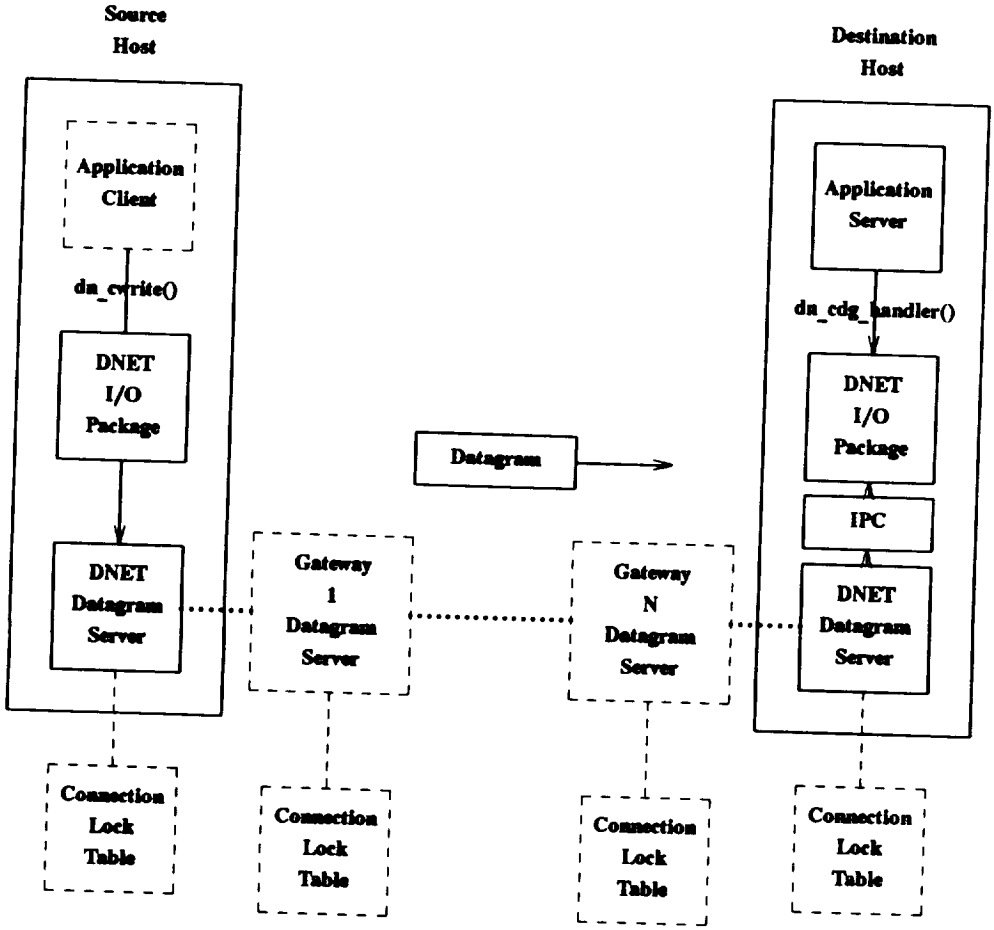
```
/*****************************************************

dechod.c

*****************************************************/
#define MAINPROGRAM

#include <stdio.h>
#include "dnet_env.h"
#include "dnet.h"
#include "dnet_errno.h"

main(argc, argv)
int         argc;
char        *argv[];
{
            int channel;
            char        debuglog[80];
            FILE        *fopen();
            extern int vms_errno;

            progname = "dechod";

            if (dn_init() < 0)
                        exit(1);

            setup_debug (debug, progname, getpid());

            if (debug)
                        fprintf(stderr,"dechod: calling dn_getclient.0);

            while ((channel = dn_getclient(progname, 0, 0)) != -1) {
                        dechod (channel);
                        dn_close(channel);
                        dn_done ();
            }
            exit(0);
}
```

```
dechod( channel)
int channel;
{
        char    buf[2*DN_BUFSIZ + 1];
        int     nrcvd, nsent;

        while (1)  {
                if ((nrcvd = dn_read(channel, buf, sizeof(buf) - 1)) <= 0){
                        if (debug && nrcvd == 0)
                                fprintf(stderr, "dechod: normal termination (%d)0, nrcvd);
                        else if (nrcvd < 0)    {
                                fprintf(stderr, "*** ERROR *** dechod: dn_read failed (%d)0,
                                        nrcvd);
                        }
                        break;
                }
                if (debug) {
                        fprintf(stderr, "dechod: %d bytes read0, nrcvd);
                        fprintf(stderr, "dechod: before dn_write0);
                }
                if ((nsent = dn_write(channel, buf, nrcvd)) <= 0){
                        fprintf(stderr, "*** ERROR *** dechod: nsent(%d) <= 00, nsent);
                        break;
                }
                if (debug)
                        fprintf(stderr, "dechod: (%d bytes) written0, nsent);
        }
}
```

5. Connectionless Mode Services

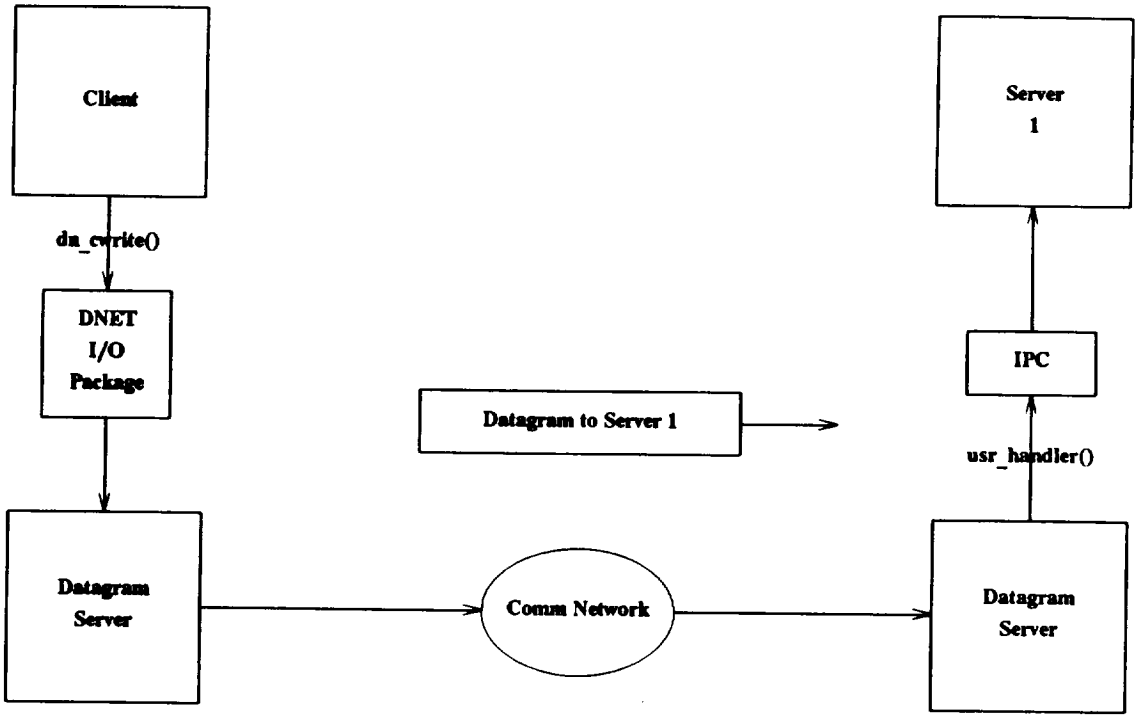
5.1 Introduction

The DNET connectionless communications service is shown schematically in the following diagram. Client and server process pairs employ the DNET BASIC I/O Library to generate datagrams which are routed automatically via DNET Datagram Servers to the destination process.



Applications using the connectionless service use only a small collection of library functions. The initiating process (process sending the datagram) invokes **dn_cwrite()**. There are two options for the receipt of datagrams, blocking and non-blocking or asynchronous reception. Synchronous datagrams are easily received by calling the function **dn_cread()**. Processes which expect to receive asynchronous datagrams, (in general, all DNET applications), must call **dn_cdg_handler()** at start up to identify an "asynchronous completion routine" to be executed whenever a connectionless datagram arrives for this

process. A completion routine must be defined which will include a call to dn_cread().



5.1.1 Connectionless Datagram Formats

The general format of a DNET connectionless datagram is:

```
struct udg
{
        struct  node src;
        struct  node next;
        struct  node dest;
        int     maxhops;
        int     type;
        long    buflen;
        char    buf[D_MAXDG];
};
```

where the struct node is defined as:

```
struct node
        char    host[I_MAXHNAME];
        char    net[I_MAXNNAME];
        char    proc[I_MAXPNAME];
```

Although the udg.buf field is defined to be D_MAXDG bytes long, the DNET will not assemble a datagram that is larger than D_MAXDG bytes long. Therefore, there are a number of bytes defined

5.2 Details of Datagram Services

5.2.1 Registration with the local Datagram Master Server

The function **dn_cinit()** must be called (after calling dn_init()) by any process which wishes to use the datagram service.

Important: **Exiting processes must call the function *dn_cdone*() in order to de-register with the Datagram Master Server; If this is not done, subsequent executions of the process will result in errors.**

5.2.2 Sending a Datagram

A process which wishes to send a datagram must include the following elements within its code.

```
{
processname = "XXXXX"

dn_init();          /* mandatory */


/* register with the DGMS
*/
dn_cinit();

/* populate src and destination of datagram
*/
udg_s.src.host = "dacvax";
udg_s.src.net = "spanet";
udg_s.src.proc = "hack_sender";

udg_s.dest.host = "dac3b2";
udg_s.dest.net = "dnett1";
udg_s.dest.proc = "hack_receiver";


/* write the datagram
*/
dn_cwrite(  );



/* De-register with the DGMS
*/
dn_cdone ();
}
```

dn_cwrite

dn_cwrite(udg_r, flags)

```
        struct udg_r *udg;
        int flags;
```

dn_cwrite() is used by DNET processes to send connectionless datagrams to other DNET processes. This operation is always synchronous.

5.2.3 Receiving Datagrams

DNET connectionless datagrams may be received both synchronously and asynchronously.

5.2.3.1 Enabling Datagram Reception Each process which needs to be able to receive connectionless datagrams must use the following format.

```
{
processname="XXXXX"

dn_init();          /* mandatory */

dn_cinit();


dn_cread(udg,flags); - may be called asynchronously using Signal or asynch read
                       routine under appropriate circumstances (see text)

dn_cdone();


}
```

5.2.3.2 Synchronous Datagram Reception

Datagrams may be received synchronously by calling **dn_cread()**. as shown in the preceding skeleton example. This call is normally 'blocking', i.e. it will not return until a datagram arrives or an error condition is detected. The function may also act in a non-blocking (or polling) fashion if the NO_WAIT flag is passed in the 'flags' argument.

```
dn_cread(udg_r, flags)
        struct udg_r *udg;
        int flags;
```

5.2.3.3 *Asynchronous Datagram Reception*

The asynchronous reception of DNET datagrams is used when the DNET process is likely to be involved in other activities when a datagram arrives. In this situation, datagrams may be viewed as 'interrupts' to the main process. Such a view requires the specification of an interrupt handler which simply describes what (software) steps should be taken when a datagram arrives. Depending on the logic, this handler may selectively accept/reject arriving datagrams.

The handler is specified by calling the function:

dn_chandler(dhandle, d_alert_sig, udg)

where dhandle()
is the interrupt handler (interrupt completion routine),

d_alert_sig is the signal
used to notify the main process of datagram arrival

and

udg points to a DNET datagram structure.

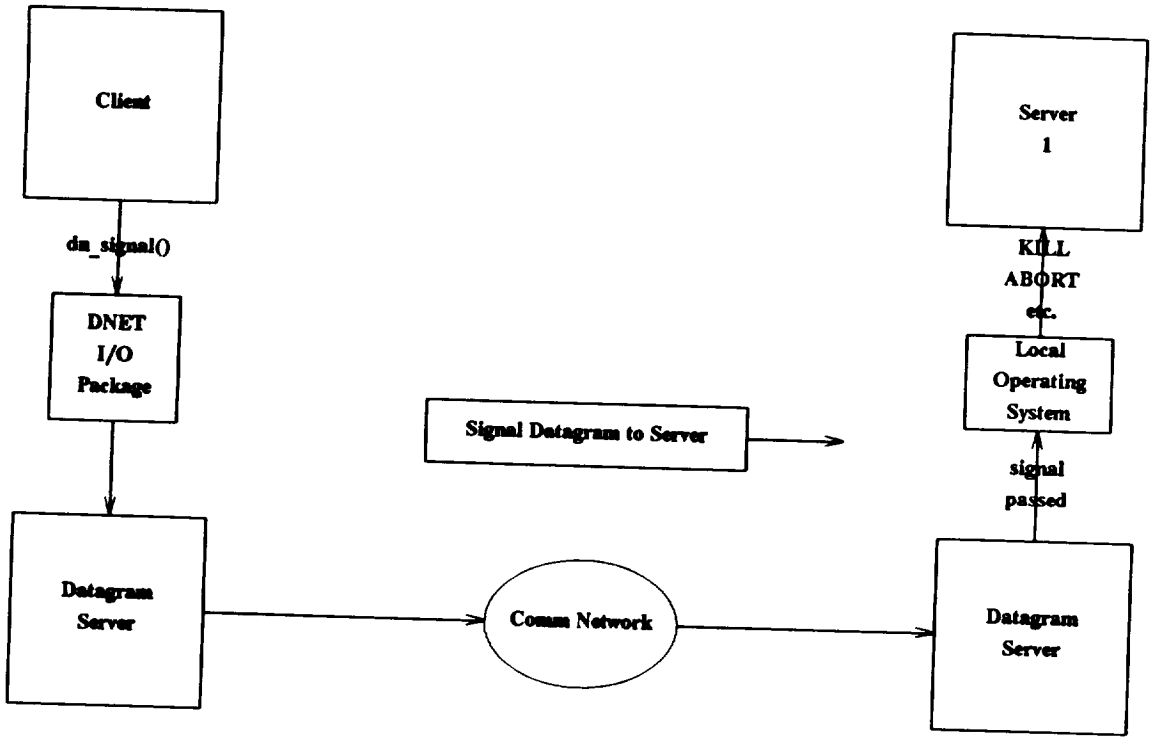
5.3 Return Receipt Service

A provision is made for return receipts for DNET datagrams. The process which sends a datagram and wishes a 'receipt' needs to set the 'return receipt' flag when calling **dn_cwrite()**.

dn_handler must read the receipt flag in datagram and return receipt to calling process via return datagram

5.4 Signalling Services

Signalling between processes is viewed as a special case of the connectionless service within DNET.



5.4.1 Sending a Signal

5.4.2 Receiving a Signal

5.5 A Connectionless Service Example

An example of the use of the connectionless service is provided in this section. The example is an elementary 'signalling' application. The client process **bcd** sends the text message "ABORT" in a connectionless datagram to a *network, host, process* specified on the command line:

bcd *network host process*

The server process **abc** is a 'trivial' process which is started, then idles waiting for an abort message from a **bcd** process.

Following is the source code for the client process.

```
/*
*       Module: bcd.c
*       Version: 1.19
*       Delta Date: 5/31/89 13:49:38
*/

#include "dnet_env.h"

#define MAINPROGRAM

#include <stdio.h>

#include "dnet.h"
#include "dnet_errno.h"

/* This redefinition of the user datagram structure in the user's
* will be replaced by providing the dn_alloc function to the user */

main (argc, argv)
int     argc;
char    *argv[];
{
        char    *getenv();
        struct udg *udg;
        static char udgbuffer[512];

        udg = (struct udg *)udgbuffer;

        if (argc != 4)  {
                fprintf(stderr, "Usage: %s destnet desthost destproc0, argv[0]);
                exit(1);
        }
```

```
        strcpy(udg->dest.net, argv[1]);
        strcpy(udg->dest.host, argv[2]);
        strcpy(udg->dest.proc, argv[3]);
        strcpy(udg->buf, "ABORT");
        udg->buflen = strlen(udg->buf) + 1;
        debug = 0;
        progname = "dmskill";
        fprintf(stderr, "dmskill: before dn_cinit0);

        if (dn_cinit(progname) = = -1)    {
                dn_cerror();
                exit(1);
        }

        fprintf(stderr, "dmskill: before dn_cwrite0);

        if (dn_cwrite(udg, 0) = = -1)       {
                dn_cerror();
                if(dn_cdone())
                        dn_cerror();
                exit(1);
        }
        if(dn_cdone() = = -1)
        {
                dn_cerror();
                exit(1);
        }
fprintf(stderr, "OK0);
        exit(0);
}
```

Note that the *network host and process* arguments are transferred from the command line into appropriate fields in the **user datagram**, udg. The "ABORT" message is placed in the datagram's buffer.

dn_cinit() is called to register with **bcd** with the Datagram master server, then dn_cwrite is called to send the datagram to its destination.

Finally dn_cdone() is called to de-register **bcd** with the Datagram Master Server.

The server process code is presented below:

```
/*
*       Module: abc.c
*       Version: 1.22
*       Delta Date: 5/31/89 13:49:36
*/

#include "dnet_env.h"

#define MAINPROGRAM

#include <stdio.h>

#ifdef DN_EUNIX
#include <signal.h>
#endif
#include "dnet.h"

#ifdef  DN_EVMS
#include <ssdef.h>
#endif
#include "dnet_errno.h"

char udgbuffer[512];
struct udg *udg;

main(argc, argv)
int     argc;
char    *argv[];
{
        int     rtncd;
        void    dghandler();

        DEpush("main");

        udg = (struct udg *)udgbuffer;
        debug = 0;
        progname = argv[0];
#ifdef DN_EUNIX
        rtncd = (int)signal(SIGCLD, SIG_IGN);
#endif
/*
        if (dn_cinit(progname) = = -1)    {
*/
fprintf(stderr,"%s: calling dn_cinit.0, progname);
        rtncd = dn_cinit("abc");
fprintf(stderr,"dn_cinit: returns.0);
fprintf(stderr,"%s: dn_cinit returns %d.0, progname, rtncd);
        if (rtncd = = -1) {
                dn_cerror();
                DEpop();
                exit(1);
        }
```

```
    fprintf(stderr,"%s: dn_cinit successful.0, progname);
            if (dn_chandler(dghandler, SIGUSR1, udg) = = -1){
                    dn_cerror();
                    if(dn_cdone() = = -1)
                            dn_cerror();
                    DEpop();
                    exit(1);
            }
    #ifdef  DN_EUNIX
            pause();
    #endif
    #ifdef  DN_EVMS
            sys_hiber();
    #endif
            if(dn_cdone() = = -1)
            {
                    dn_cerror();
                    DEpop();
                    exit(1);
            }
            fprintf(stderr, "%s: exiting0, progname);
            DEpop();
            exit(0);
    }


    void dghandler()
    {

            DEpush("dghandler");

            fprintf(stderr, "in dghandler0);
            if (!strcmp(udg->buf,"ABORT"))
                    fprintf(stderr, "Received ABORT0);
    #ifdef  DN_EVMS
            if(sys_wake(0) = = -1)
                    fprintf(stderr, "%s: Can't wake up ./n", progname);
    #endif
            DEpop();
            return;
    }
```

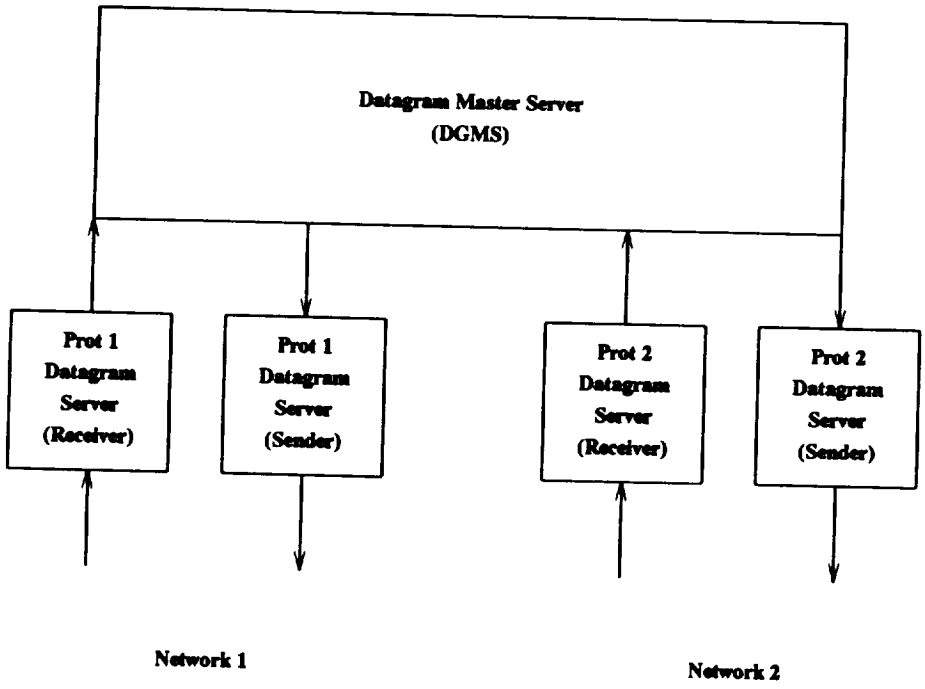
Salient points in this code include:

1. dn_cinit is called to register **abc** with the Datagram Master Server
2. dn_chandler is called; the signal SIGUSR1 is specified as the asynchronous signalling mechanism indicating arrival of datagram destined for **abc**, a completion routine *dghandler*() is specified for execution when this signal is received, and the datagram **udg** is specified as the target for incoming datagrams.
3. The process 'idles' using pause() or sys_hiber() until a datagram is received
4. dn_cdone is called to de-register with the Datagram Master Server prior to exiting

The completion routine *dghandler()* is called when the DGMS signals abc that a datagram has arrived. The buffer of udg is checked for the "ABORT" message If this message is received, the handler wakes up the idling main process, causing it to exit.

5.5.1 Datagram Protocol Servers

Datagram protocol servers DGS) are DNET processes located at each DNET host which propagate DNET datagrams through the heterogeneous network. These servers provide a network protocol specific interface between the Datagram Master Server (DGMS) and the underlying network(s). An overview of a the relationships between the DGMS and DGS's is provided in the following diagram:



5.5.1.1 Connection Lock Table

The datagram servers keep track of a pool of open connections to other DNET hosts over which connectionless datagrams may be routed. This information is contained in the Connection Lock Table

The connection lock table contains information about the hosts to which the local host has connections. It includes the protocol and logical channel number used by the BASIC I/O package in transmitting datagrams to that host.

Connection Lock Table					
Connection Owner	Stream ID	Net	Host	Proc Name	Channel #
FXFR1-Client	1	D	4	FXFR6	89988419
FXFR1-Client	2	Q	3	FXFR4	89988419
FXFR1-Client	3	T	3	FXFR2	89988419
—	—	—	—	—	—
—	—	—	—	—	—
—	—	—	—	—	—
—	—	—	—	—	—

5.6 Signalling

DNET processes may send signals to other processes within DNET by calling the function dn_signal().

5.6.1 Sending Signals

dn_signal

```
status = dn_signal(net, host, service, signal)
        int status;
        int signal;
        char *net;
        char *host;
        char *service;

        dn_signal sends a signal datagram to a server on a specified host.
```

5.6.2 Delivery of Signals

DNET signals are sent to the Datagram Server at the destination host. The Datagram Server recognizes the type SIGNAL and forwards the appropriate information to the local operating system for action. The operating system will complete actions such as 'killing' a process, etc. See the Chapter on Connectionless Service for more detailed information.

6. DNET Error Handling

DNET Basic I/O Library functions return a non-selective error code if an error is detected during their operation:

DNET applications which wish to use the error handling facility should include the header file *dnet_errno.h* and follow the procedure below:

Errors detected by the DNET code are identified in the variable **dnet_errno**:

```
dnet_errno = XXXXX;
```

An error function, **dnet_error("string")**, is then optionally called where string is an optional, user provided informative message. **dnet_error** provides detailed information on conditions when the error was detected including a stack trace.

```
dnet_error(*error_string)

char * error_string;
```

Detailed error codes are provided in an Appendix to this Guide.

7. Routing

DNET employs a hierarchecal routing scheme. Each DNET node has a routing table which lists the next DNET node to contact in order to reach each known network within DNET. A 'null' entry for the destination indicates that the local host is directly connected to the destination network. The routing tables are 'static' in the initial version of DNET, but could be easily updated via an appropriate protocol in a latter version.

The details of DNET routing operations are hidden from 'ordinary' applications and hence will not be of particular use to the application programmer.

The router selects the host/process to which the datagram will be transmitted next by calling the function **get_path**();

path = get_path(src_net,src_host,dest_net,dest_host,dest_process,numhops);

src_net is the network in which the destination host is located

src_host is the destination host

dest_net is the network in which the destination host is located

dest_host is the destination host

dest_process is the destination process

numhops - number of hops from current location to destination

Details on routing within DNET are found in the Adminstrator and Technical Guides.

8. Interprocess Communication

As part of its internal design, DNET provides a generalized interprocess communication facility. A brief description of this facility is provided here. Complete details on the use of this facility are given in the DNET Technical Guide and Reference.

9. Presentation Layer Services

DNET will provide a limited presentation layer facility.

Within the DAVID environment, the single most important coding problem across heterogeneous machines is the internal representation of data. Information moved from one machine to another may only be viewed consistently if data types are faithfully "mapped" between machines.

Thus, if the transmitting machine views integers as 32 bit quantities and represents floating point numbers with 64 bits while the receiver represents these two data types as 64 and 48 bit quantities, respectively, serious misalignment of data files will occur.

The Presentation Layer Service to be provided by DNET will be limited to a subset of the SUN (XDR) External Data Representation Protocol and the existing DAVID Presentation Services.

9.1 XDR

A subset of the SUN Microsystems External Data Representation (XDR) protocol is provided with DNET.

XDR allows arbitrary C data elements to be written and read in a consistent and accurate manner, independent of the representation of these data elements on the source/destination computers. XDR provides for the translation (or encoding) of data elements into a canonical representations at the source machine. These canonical forms may then be interpreted (or decoded) according to appropriate conventions at the destination machine. Inter-computer differences such as the number of bits and/or the byte ordering of specific data types are conveniently avoided via judicious use of XDR.

Typical XDR library functions include filter routines for strings (null terminated arrays of bytes), structures, unions, and arrays as well as primitive routines for most common data types. These filter routines are used for both encoding and decoding of the XDR canonical data stream. The encode/decode 'direction' is indicated via a flag when the filters are invoked.

Data may be encoded/decoded source/destination data "stream". This stream may be a file, a memory array, of a memory block.

9.1.1 Issues in the Use of XDR

The files **../common/dnxdr.c** and **../common/dnxdr.h** contain the XDR functions available for use within DNET. The reader is referred to the source code for additional details on the various issues discussed here.

The general procedures used for encoding/decoding of data with XDR are as follows:

1. Specification of the XDR 'handle'

2. Creation of I/O Data Stream
3. Encoding/Decoding of Data using XDR Library functions

9.1.2 *The XDR Handle (Control Structure)*

A common structure is used to 'control' the XDR operations on a particular data stream. This structure is shown below:

```
/* The XDR handle.
*/
typedef struct {
        enum xdr_op     x_op;    /* operation; fast additional param */
        struct xdr_ops {
                int (*x_getlong)();         /* get a long from underlying stream */
                int (*x_putlong)();         /* put a long to " */
                int (*x_getbytes)();        /* get some bytes from " */
                int (*x_putbytes)();        /* put some bytes to " */
                uint (*x_getpostn)();       /* get byte offset from beginning */
                bool_t (*x_setpostn)();     /* reposition loc in the stream*/
                long * (*x_inline)();       /* put some bytes to " */
                void (*x_destroy)();        /* free privates of this xdr stream */
        } *x_ops;
        caddr_t x_public;           /* users' data */
        caddr_t x_private;          /* pointer to private data */
        caddr_t x_base; /* private used for position info */
        int     x_handy;            /* extra private word */
        int     x_size;  /* yet another */
} XDR;
```

9.1.3 *Creation of the I/O Datastream*

Two functions may be used to create the XDR datastream. The function used depends on whether the stream is to be a file or an area of memory.

File

```
xdrstdio_create(xdrs, fp, x_op)
        XDR *xdrs;
        FILE *fp;
        enum xdr_op x_op;
```

x_op is chosen from among:

```
        XDR_ENCODE
        XDR_DECODE
```

Memory

xdrmem_create(xdrs, addr, len, x_op)

XDR *xdrs;
char *addr;
u_int len;
enum xdr_op x_op;

x_op is chosen from among:

XDR_ENCODE
XDR_DECODE

9.1.4 Encoding/Decoding of Data using XDR library

9.1.4.1 Primitive Filters Example of a typical primitive:

bool_t xdr_xxx(xdrs, fp) XDR *xdrs; xxx *fp; {

}

Comments:

xdrs points to the XDR control structure

fp points to the data stream

ENCODE/DECODE already specified in the XDR control structure

Returns TRUE (1) if successful
Returns FALSE (0) if failure

The primitive names are usually adequate to describe the data type involved. e.g :

xdr_long(&xdrs, stdin)

9.1.4.2 Non-filter Primitives

Two especially useful ancillary functions allow determining or setting the current position in the XDR datastream.

Get current position in the datastream

u_int xdr_gettpos(xdrs, pos)
XDR *xdrs;
u_int pos;

Set current position

```
bool_t xdr_setpos(xdrs, pos)
        XDR *xdrs;
        u_int pos;
```

9.1.4.3 Higher Level Filters Arrays

9.1.4.4 An Introductory Example

Consider the following simple example. The name of a file and its size, in bytes, is to be passed between machines with consistent interpretation.

We define a structure in which to place the file information and assume that some convenient utility is used to populate this structure.

```
struct
{
        char[100]       filename;
        long    filesize;
} filedescrip;

bool_t xdr_filedescription(xdrs, filedes)
XDR *xdrs;
struct filedescrip *filedes;
{
        return( xdr_stream(xdrs, &filedes->filename) &&
                xdr_long(xdrs, &filedes->filesize) );
}
```

Source:

```
/* Declare an instance of the XDR 'handle'
*/
XDR xdrs;

/* open the 'Canonical' File
*/
fp = fopen ("Fileinfo", "w");

/* Setup the xdr handle to point to 'Fileinfo' and to encode the
  datastream
*/
xdrstdio_create (&xdrs, fp, XDR_ENCODE);

/* Encode the file information (from file struct) into the open datastream
*/
filedescription(&xdrs, fp);,

/* Close the file
*/
fclose(fp);

/* Send the file to its destination using a convenient function
*/
put_file();
```

```
Destination:

/* Declare an instance of the XDR 'handle'
*/
XDR xdrs;

/* Receive the file at its destination using a convenient function
*/
receive_file();

/* open the 'Canonical' File
*/
fp = fopen ("FileInfo", "w");

/* Setup the xdr handle to point to 'FileInfo' and to decode the
  datastream
*/
xdrstdio_create (&xdrs, fp, XDR_DECODE);

/* Encode the file information (from file struct) into the open datastream
*/
filedescription(&xdrs, fp);,

/* Close the file
*/
fclose(fp);
```

9.1.5 Example - use of XDR in dnetstat

We next consider an example drawn from the actual DNET implementation. The DNET client/server pair dnetstat and dnstatd use XDR in order to accurately pass DNET status structures across the heterogeneous network.

The 'standard' XDR library function, xdrmem_create(), is used to create a data stream at a specific location in memory, in this case in the data buffer of a DNET connectionless datagram which is being assembled for shipment to the **dnetstat** client.

The steps performed are:

1. Populate the data structure for the network status
2. Create a temporary memory area (in the buffer for a DNET connectionless datagram
3. Invoke a function which encodes/decodes the data structure to/from XDR format

```
struct udg udg_s;
struct dmsinfo dms_stat;

main()
{
        XDR xdrs

        xdrmem_create(&xdrs, udg_s->buf, sizeof (udg_s->buf), XDR_ENCODE);
        xdr_sit_instance(&xdrs,&dms_stat);
}
```

Discussion

xdrmem_create() is set up to place the XDR encoded structure in the data buffer (usg_s->buf) of a DNET connectionless datagram which is being assembled in memory for shipment to some remote destination.

Since **dnetstat** is a designed to be a currently used DNET application, its accurate interchange of information warrants special attention. The functions **xdr_sit_instance()** and **xdr_adgut_instance()** were 'custom' written for this purpose. These functions and the data structures which they Encode/Decode are presented below:

```
struct ms_entry {
        char    service[80];
        char    image[80];
        int     prespawned;
        int     max;
        int     avail;
        int     inuse;
        int     seqno;
        struct si_entry *si_table;
};      /* for generic table */
struct msinfo {
        char    service[80];
        char    image[80];
        int     prespawned;
        int     max;
        int     avail;
        int     inuse;
        int     seqno;
};      /* for generic table */
```

```
struct si_entry {
        int     pid;
        int     inuse;
        int     inited;
        long    stime;
        struct ms_entry *ms_entry;
        int     term_sent;
        int     pending;
        int     buflen;
        char buf[BUFSIZ];
};      /* for instance table */

struct siinfo {
        int     pid;
        int     inuse;
        int     inited;
        long    stime;
        int     term_sent;
        int     pending;
};      /* for instance table */

typedef struct ms_entry MS_ENTRY;
typedef struct si_entry SI_ENTRY;

#define DMS_GETCLIENT 1
#define DMS_GETSTATUS 2

struct dms_request  {
        int pid;
        int type;
};

#define DMSTAT_END          0
#define DMS_INFO            1
#define DMSTAT_INFO         2

struct dmsinfo  {
        int type;           /*  DMSTAT_INFO, DMSTAT_END */
        int numsi;
        struct msinfo ms;
        struct siinfo si[100];
};
```

```
/* structure for ADGUT (connectionless service) entry
*/
struct dgms_adut
{
        int pid;  /* Process Identifier */
        char pname[D_MAXPNAME];    /* Process name bound to */
        char ipcname[D_MAXPATHNAME];/* Name of IPC mechansim for sending */
        int ipcid;          /* ipcid of ipcname */
        int maxmsg;         /* Maximum number of bytes that this user can handle */
        int signal;         /* Signal number used to inform of impending message */
        unsigned w_timeout;       /* Timeout period on write */
        time_t add_time;/* Time adgut entry was added */
        time_t last_access;         /* Time adgut entry was last accessed */
        time_t last_update;         /* Time adgut entry was last updated */
        time_t last_send;           /* Time last datagram was sent to this process */
        time_t last_recv; /* Time last datagram was received from this process */
        int state;          /* 0 - Invalid
                                   1 - Basic
                                   2 - Listen */
};
```

```
/* dnetstat utilities - User Network Status Function
*/

#include <stdio.h>
#include <ctype.h>

#include "dnet_env.h"

#include <signal.h>

#ifdef DN_EUNIX
#include <fcntl.h>

#ifdef DN3B2
#include "/usr/netinclude/sys/time.h"
#else
#include <sys/time.h>
#endif

#endif


#ifdef DN_EVMS
#ifdef DNDFTNIC
#include time
#else
#include "time.h"
#endif
#endif

#include "dnet.h"
#include "dnet_errno.h"
#include "dnet_ipc.h"
#include "dgms.h"
#include "dms.h"
#include "dnetstat.h"
#include "dnxdr.h"
```

```
/* Connection Service Definitions
*/

xdr_sit_instance(xdrs,dms_bptr)
XDR *xdrs;
struct dmsinfo *dms_bptr;
{
        char *cpp;
        int i;

        if (!xdr_int(xdrs,&dms_bptr->type) )
                return(FALSE);
        if (!xdr_int(xdrs,&dms_bptr->numsi) )
                return(FALSE);

        cpp = dms_bptr->ms.service;
        if (!xdr_string(xdrs,&cpp,D_MAXPATHNAME))
                return(FALSE);

        cpp = dms_bptr->ms.image;
        if (!xdr_string(xdrs,&cpp,D_MAXPATHNAME))
                return(FALSE);

        if (!xdr_int(xdrs,&dms_bptr->ms.prespawned) )
                return(FALSE);
        if (!xdr_int(xdrs,&dms_bptr->ms.max) )
                return(FALSE);
        if (!xdr_int(xdrs,&dms_bptr->ms.avail) )
                return(FALSE);
        if (!xdr_int(xdrs,&dms_bptr->ms.inuse) )
                return(FALSE);
        if (!xdr_int(xdrs,&dms_bptr->ms.seqno) )
                return(FALSE);

        if (debug)
         fprintf(stderr,"xdr_sit_instance: numsi = %d0,dms_bptr->numsi);

        for (i=0; i< dms_bptr->numsi; i++) {
         if (!xdr_int(xdrs,&dms_bptr->si[i].pid) )
                return(FALSE);
         if (!xdr_int(xdrs,&dms_bptr->si[i].inuse) )
                return(FALSE);
         if (!xdr_int(xdrs,&dms_bptr->si[i].inited) )
                return(FALSE);
         if (!xdr_int(xdrs,&dms_bptr->si[i].stime) )
                return(FALSE);
         if (!xdr_int(xdrs,&dms_bptr->si[i].term_sent) )
                return(FALSE);
         if (!xdr_int(xdrs,&dms_bptr->si[i].pending) )
                return(FALSE);
        }

        return(TRUE);
}
```

```
xdr_adgut_instance(xdrs,adg_bptr)
XDR *xdrs;
struct dgms_adut *adg_bptr;
{
        char *cpp;

        if (!xdr_int(xdrs,&adg_bptr->pid) )
                return(FALSE);

        cpp = adg_bptr->pname;
        if (!xdr_string(xdrs,&cpp,D_MAXPATHNAME))
                return(FALSE);

        cpp = adg_bptr->ipcname;
        if (!xdr_string(xdrs,&cpp,D_MAXPATHNAME))
                return(FALSE);

        if (!xdr_int(xdrs,&adg_bptr->ipcid) )
                return(FALSE);
        if (!xdr_int(xdrs,&adg_bptr->maxmsg) )
                return(FALSE);
        if (!xdr_int(xdrs,&adg_bptr->signal) )
                return(FALSE);
        if (!xdr_u_int(xdrs,&adg_bptr->w_timeout) )
                return(FALSE);
        if (!xdr_int(xdrs,&adg_bptr->add_time) )
                return(FALSE);
        if (!xdr_long(xdrs,&adg_bptr->last_access) )
                return(FALSE);
        if (!xdr_long(xdrs,&adg_bptr->last_update) )
                return(FALSE);
        if (!xdr_long(xdrs,&adg_bptr->last_send) )
                return(FALSE);
        if (!xdr_long(xdrs,&adg_bptr->last_recv) )
                return(FALSE);
        if (!xdr_int(xdrs,&adg_bptr->state) )
                return(FALSE);

        return(TRUE);
}
```

9.2 Transferring arbitrary files using XDR

No supporting mechanisms are currently offered in DNET for the problem of transferring arbitrary files using XDR.

9.3 Existing DAVID Presentation Service

Existing DAVID system currently includes a pair of data conversion functions which map data types into a straightforward, virtual format for interchange with machines employing different internal representation.

9.3.1 Virtual Data Format for DNET Transmission

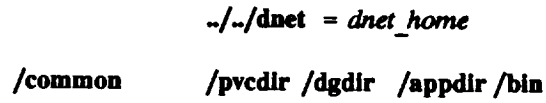
ASCII representation

```
COMP_STATUS dcp_tpack(vca,cca,nvisit,tca,ptca,fptr)
  VCA *vca;
  CCA *cca;
  USHORT *nvisit;
  TCA *tca;
  TCA *ptca;
  FILE *fptr;


COMP_STATUS dcp_tupack(vca,cca,fptr,ptrfile)
  VCA *vca;
  CCA *cca;
  FILE *fptr;
  FILE *ptrfile;
```

10. Standard DNET Code Organization

10.1 Standard Directory Structure

The 'standard' DNET directory structure is shown in the figure below:



NOTE: Programming tasks covered by this Guide should generally require modifications to files in ../dnet/ and ../dnet/appdir

Changes to the subdirectories ../common, ../dnet/pvcdir, ../dnet/dgdir should only be undertaken with a view toward global changes in mind.

10.2 Variation for VMS Installations

The DNET directory-tree on VMS systems is logically identical to that on UNIX systems. It differs only in the syntax used to reference directories:

dnet_home:[.common]
dnet_home:[.pvcdir]
dnet_home:[.dgdir]
dnet_home:[.appdir]
dnet_home:[.bin]

11. Compiling & Making DNET Applications Programs

11.1 General Strategy

Use existing DNET applications as a model for **make** files

The relevant libraries in **dnet_home** directory are placed in the *dnet_home* directory.

11.2 Setting DNET Compile Time Environment Variables

These Environment variables are ordinarily set automatically based on the machine name provided to the DNET **postmove** utility program. Typical of the environment to be specified are:

1. Communication Protocol(s)
2. TCP/IP Implementation
3. Target Machine Type
4. Target Operating System

The most convenient means of setting these variables is to create an entry for the target DNET machine in the file *dnet_home/tbls.db*. This is a database file which contains all relevant information about the target node.

11.3 Making UNIX Version

1. cd **dnet_home**
2. make

11.3.1 BSD Systems

Special considerations - Must run 'ranlib' manually on the libaries generated during the 'make' procedure.

ranlib dnet.a

ranlib dnettcp.a

This may be accomplished by running 'make' twice on the target machine; this has the effect of running ranlib twice.

11.3.2 Example Make File

A typical UNIX makefile is show below. This file is used to make the DNET application files. All relevant makefiles are presented in the source code listings.

```
	$(CC) -c $(CFLAGS) $<
	$(AR) $(ARFLAGS) $@ $*.o
	rm -f $*.o

	$(GET) $(GFLAGS) $<
	$(CC) -c $(CFLAGS) $*.c
	$(AR) $(ARFLAGS) $@ $*.o
	rm -f $*.[co]

DNET=../dnet.a
DNETTCP=../dnettcp.a
DNETDEC=../dnetdec.a
CDIR=../common
BIN=../bin
#DNETDG=../dgdir/libdn_dg.a
HEAD=$(CDIR)/dnet.h $(CDIR)/dnet_env.h $(CDIR)/dnet_errno.h $(CDIR)/dnet_ipc.h $(CDIR)/dnxdr.h
AR=ar
ARFLAGS=rv
CFLAGS=-g -I$(CDIR) -DDN3B2 -DDN_ETCP
CCLINK=cc $(CFLAGS) -o $@ $@.c $(LIBS)
WOOL=/usr/lib/libnet.a /usr/lib/libnsl_s.a
LIBS=$(DNETTCP) $(DNET) $(WOOL)

all:		echo mskill netstat rexec tftp ncl login mail

mail:		$(BIN)/dmail $(BIN)/dmaild $(BIN)/checkdmail

ncl:		$(BIN)/dncl $(BIN)/dncld $(BIN)/dncld_unix

echo:		$(BIN)/decho $(BIN)/dechod $(BIN)/dechon

login:		$(BIN)/dlogin $(BIN)/dlogind

mskill:		$(BIN)/dmskill

netstat: $(BIN)/dnetstat $(BIN)/dnstatd

rexec:		$(BIN)/drexec $(BIN)/drexecd

tftp:		$(BIN)/dtftp $(BIN)/dtftpd
```

```
$(BIN)/decho:   decho.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/decho decho.o $(LIBS)

$(BIN)/dechod: dechod.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/dechod dechod.o $(LIBS)

$(BIN)/dechon: dechon.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/dechon dechon.o $(LIBS)

$(BIN)/dmskill: dmskill.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/dmskill dmskill.o $(LIBS)

$(BIN)/dnetstat: dnetstat.o dnstatutil.o $(DNET) $(DNETTCP) $(CDIR)/dgms.h $(CDIR)/dms.h dnetstat.h $(
                        cc -o $(BIN)/dnetstat dnetstat.o dnstatutil.o $(LIBS)

$(BIN)/dnstatd: dnstatd.o dnstatutil.o $(DNET) $(DNETTCP) $(CDIR)/dgms.h $(CDIR)/dms.h dnetstat.h $(C
                        cc -o $(BIN)/dnstatd dnstatd.o dnstatutil.o $(LIBS)

$(BIN)/drexec:  drexec.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/drexec drexec.o $(LIBS)

$(BIN)/drexecd: drexecd.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/drexecd drexecd.o $(LIBS)

$(BIN)/dtftp:   dtftp.o dtftputil.o dnlog.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/dtftp dtftp.o dtftputil.o dnlog.o $(LIBS)

$(BIN)/dtftpd:  dtftpd.o dtftputil.o dnlog.o $(DNET) $(DNETTCP)
                        cc -o $(BIN)/dtftpd dtftpd.o dtftputil.o dnlog.o $(LIBS)
```

```
$(BIN)/dpresent:        dpresent.o $(DNET) $(DNETTCP) $(CDIR)/dgms.h $(CDIR)/dms.h dpresent.h
                                cc -o $(BIN)/dpresent dpresent.o $(LIBS)
$(BIN)/dmail:   dmail.o sendmail.o readmail.o dtftputil.o $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/d
                cc -o $(BIN)/dmail dmail.o sendmail.o readmail.o dtftputil.o $(LIBS) -lcurses -ltermcap

$(BIN)/dmaild:  dmaild.o dtftputil.o    $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/dms.h $(CDIR)/dn
                cc -o $(BIN)/dmaild dmaild.o dtftputil.o $(LIBS)

$(BIN)/checkdmail:      checkdmail.o    $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/dms.h $(CDIR)/dn
                cc -o $(BIN)/checkdmail checkdmail.o $(LIBS)

$(BIN)/dncl:    dncl.o dncl_utils.o $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/dms.h $(CDIR)/dnet_err
                cc -o $(BIN)/dncl dncl.o dncl_utils.o $(LIBS)

$(BIN)/dncld:   dncld.o dncl_utils.o $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/dms.h $(CDIR)/dnet_er
                cc -o $(BIN)/dncld dncld.o dncl_utils.o $(LIBS)

$(BIN)/dncld_unix:      dncld_unix.o dncl_utils.o $(DNET) $(DNETTCP) $(CDIR)/dnet.h $(CDIR)/dms.h $(C
                cc -o $(BIN)/dncld_unix dncld_unix.o dncl_utils.o $(LIBS)

$(BIN)/dlogin:  dlogin.o dlogutils.o $(DNET) $(DNETTCP)
                                cc -o $(BIN)/dlogin dlogin.o dlogutils.o $(LIBS)

$(BIN)/dlogind: dlogind.o dlogutils.o $(DNET) $(DNETTCP)
                                cc -o $(BIN)/dlogind dlogind.o dlogutils.o $(LIBS)


decho.o:        $(HEAD)
dechod.o:               $(HEAD)
dechon.o:               $(HEAD)
dmail.o:        $(HEAD) dmail.h
dmai                    $(HEAD) dmail.h
dmsk                    $(HEAD)
dnetstat.o:             $(HEAD) dnetstat.h
dnstatd.o:              $(HEAD) dnetstat.h
dnstatutil.o:           $(HEAD) dnetstat.h
drexec.o:               $(HEAD)
drexecd.o:              $(HEAD)
dtftp.o:        $(HEAD) dtftp.h
dtftpd.o:               $(HEAD) dtftp.h
dtftputil.o:            $(HEAD) dtftp.h
dnlog.o:        $(HEAD)
dpresent.o:             $(HEAD)
dlogin.o:               $(HEAD)
dlogind.o:              $(HEAD)
dlogutils.o:            $(HEAD)
dncl.o:         $(HEAD) dncl.h
dncld.o:        $(HEAD) dncl.h
dncld_unix.o:           $(HEAD) dncl.h
dncl_utils.o:           $(HEAD) dncl.h
```

11.4 Making VMS Version

11.4.1 General

DNET currently employs VMS 'command' files as a pseudo 'make' facility. These files are simply scripts for executing the various steps necessary to 'make' DNET on the target VAX machine. Since this is not a true make facility, these files DO NOT check for the date of executables versus source files, requiring instead that the user keep track of incremental changes in the source code and the 'side' effects of these changes on the several executables.

11.4.2 MicroVAX II

1. cd **dnet_home**
2. @make.dv

$ define c$include dnet_common, dnet_pvcdir, dnet_dgdir, wool_sys, -
wool_netinet $ define vaxc$include c$include, sys$library $ lib/create dnet $ lib/create dnetdec $ lib/create dnettcp $ cd [.common] $ @decnet.m $ @tcp.m $ @vms.m $ cd [-.pvcdir] $ @dnet.m $ @drelay.m $ @dms.m $ cd [-.dgdir] $ @mail $ cd [-.appdir] $ @decho.m $ @dechon.m $ @drexec.m $ @dtftp.m $ @dmskill.m $ @dnstat.m $ @dlogin.m $ @dncl.m $ @dmail.m $ cd [-.bin] $ purge *.exe $ $ cd dnet_home $

The makefile for *decho* is presented below as a representative example of making a specific application.

11.4.3 NASA-GSFC VAXes (IAF, DFTNIC, etc. using Excelan TCP)

Enter the following commands

1. cd **dnet_home**
2. @make.dft

11.5 Making individual files

It is obviously possible to make individual files via manual steps or via selective 'makes' of either the *common, pvcdir, dgdir, or appdir* makefiles. It is important to note that there are numerous interactions between the 'core' DNET files in *common, pvcdir, and dgdir*. Any changes to these files may have wide ramifications and considerable functional testing of all DNET operations is advised after such tests.

DNET applications which follow the basic rules in this GUIDE are more 'self-contained' and may usually be altered without significant effect on other applications.

12. Debugging DNET applications

For convenience, a generalized 'logging' facility is provided in order to allow a 1st order indication of DNET operations. This may be used as a 'debugging' aid when problems arise with DNET and the user is unfamiliar with the specific debugging tools on the local machine..

This facility is activated when the DNET "environment variable" is set. This varies with the operating system as follows:

1. UNIX

   shell dnet_debug=1;export dnet_debug

   C shell setenv dnet_debug 1

2. VMS

   define/job dnet_debug 1

The debugging files will be placed in the directory and named as follows:

Directories:

UNIX

/tmp/dnet

VMS dnet_home

The log files are generated for each DNET server process (most clients will 'dump' messages to the terminal instead of a file) and are named as follows:

XXX###.log

where XXX is the process name

and ### is the process ID

UNIX files may be viewed while DNET is operational using

On VMS systems, DNET must be stopped before the log files may be inspected.

NOTE:

Care should be exercised in the use of this debugging technique as log files of considerable size may be generated over time. Thus the 'debug' option should only be activated long enough to study a problem, then deactivated by setting *dnet_debug = 0*

13. DNET Error Codes

```
#define D_NOERR      0    /* No DNET error */
#define D_SYSERR     1    /* A system error has occurred */
#define D_BADSTATE   2    /* program in wrong state to issue this dnet call */
#define D_BADARG     3    /* value of argument was determined to be invalid */
#define D_OVRFLW     4    /* overflow of i/o buffer */
#define D_AEXIST     5    /* The specified object already exists */
#define D_ESRVRSP    6    /* Error return value in DGMS service req response */
#define D_EPERM      7    /* Permission Denied */
#define D_NOMSG      8    /* D_NOWAIT flag set and no message waiting to be read */
#define D_NODGRSC    9    /* No more available DGMS resources */
#define D_INTERN     10   /* Internal DNET error */
#define D_BADNM      11   /* Invalid process name was specified */
#define D_DGTB       12   /* Datagram To Big */
#define D_MSGTB      13   /* Message To Big */
#define D_BADHN      14   /* Could not find net/host combination in router tables */
#define D_ADGENF     15   /* ADGUT Entry Not Found */
#define D_PN2BIG     16   /* Process name string too big */
#define D_IPCNM2BIG  17   /* IPC name string too big. DNET code error */
#define D_NOEXIST    18   /* The specified object does not exist */
#define D_INTR       19   /* A signal interrupted the library routine */
#define D_NOSRSC     20   /* Temporarily out of system resources */
#define D_NODNET     21   /* Missing all or part of dnet provider */
#define D_WOULDBLOCK 22       /* Operation would block */
#define D_TIMEOUT    23   /* Timeout or retry count exceeded */
#define D_QUOTA      24   /* Quota limit exceeded */
#define D_NOSYSFILE  25   /* DNET system file/table not found */
#define D_SYNERR     26   /* DNET system file/table syntax error */
#define D_NOIMAGE    27   /* Image (server) not file not found */
#define D_HOMELESS   28   /* Env variable 'dnet_home' not defined */
#define D_SRVNOACK   29   /* No respone from application server */
#define D_NOHOST     30   /* No such host */
#define D_NOPATH     31   /* DNET could not find a path for the src/dest pair */
#define D_SYSLIBERR  32   /* System library function failed */
#define D_NODNETSRV  33       /* DNET servers dms/dgstcp not defined in 'etc/services'*/
#define D_SHUTDOWN   34   /* Orderly shutdown from master server */
#define D_MAXERRS    35
```

```
static   char    *dgms_errmsgs[D_MAXERRS] = {
"No DNET error",
"A system error has occurred",
"program in wrong state to issue this dnet call",
"value of argument was determined to be invalid",
"overflow of i/o buffer",
"The specified object already exists",
"Error return value in DGMS service req response",
"Permission Denied",
"D_NOWAIT flag set and no message waiting to be read",
"No more available DGMS resources",
"Internal DNET error",
"Invalid process name was specified",
"Datagram To Big",
"Message To Big",
"Could not find net/host combination in router tables",
"ADGUT Entry Not Found",
"Process name string too big",
"IPC name string too big. Probably DNET internal code error",
"The specified object does not exist",
"A signal interrupted the library routine",
"Temporarily out of system resources",
"Missing all or part of dnet provider",
"Operation would block",
"Timeout or retry count exceeded",
"Quota limit exceeded",
"DNET system file/table not found",
"DNET system file/table syntax error",
"Image (server) file not found",
"Env variable 'dnethome' not defined",
"No respone from application server",
"No such host",
"DNET could not find a path for the src/dest pair",
"System library function failed",
"DNET servers dms/dgstcp not defined in '/etc/services'",
"Orderly Shutdown from master server"
};
```

14. Glossary

The following terms are used in the description of DNET:

Applications Servers-

Servers such as File Transfer, Remote Login, Remote Execution, etc. that perform services for clients. Applications Servers are invoked on demand by clients after using the Service Assignment to obtain the name of an available server.

Connection Lock Table-

List of open connections held by process for use by its Basic Datagram I/O package. Locked connections result from user requests for Permanent Virtual Circuits.

Datagram Master Server (DGMS)-

A server process, located at each DNET host and gateway, which provides an interface to DNET clients and servers and the DNET Connectionless Datagram and Signalling Service

Datagram Protocol Servers (DPS)-

Protocol specific servers located at each DNET host and gateway, which provides an DNET Connectionless an interface to the underlying network Datagram service.

Master Server Init Table-

These tables, **tbls.msinittcp** and **tbls.msinitdec** contain initialization information for the DNET Master Servers including the type of server to be activated, the maximum # allowed at this host, and the number to make available initially, and an indication of whether the server must be prespawned. The tables are updated by the local System Administrator at the specific DNET host.

Master Server Table-

One for each DNET host, it contains information on the types and numbers of each class of DNET server actively supported on this node at any instant. Each generic server entry points to a **Server Instance Table** which lists the current specific instances of a particular class of server. It is updated by the Master Server and by specific DNET application servers.

Master Server Process (DMS)-

Processes, one per Network, managing the Master Server Table, handling server registration, server assignment, and server control. They are spawned by network start-up command files.

DNET Basic I/O package-

Included as library within an application program, it provides network i/o interface including datagram formatting.

Gateway-

A DNET node at which communicaton protocol boundary is passed. DNET relay servers move data from one network to another performing an effective protocol conversion for streaming services. These servers are created, allocated, and used like any other DNET streaming applications servers. The Datagram Master Server, in conjunction with protocol specific datagram servers performs a similar function for DNET datagrams.

Network Command Line Interpreter-

DNET Client process that directs the execution of network commands using datagrams sent to various hosts and several servers.

myname - hostname table-

A table, **tbls.myname**, maintained in the **dnet_home** directory on each DNET node lists the DNET networks to which that host is connected and the name(s) by which the local host is known on those networks.

Network Command Language Processor-

Server that directs the execution of network commands using datagrams sent to various hosts and several servers. It is an application server, copies can be pre-spawned or spawned on demand.

Network Command Server-

Spawned by request from Command Language Processor, this Server is directed by Command Language Processor. It spawns processes and directs i/o to execute network commands.

Network Status Server-

Resides in each network host. Receives Host Status Tables, Host Alias Table, Well Known Server Tables, Connectivity Tables, and periodically sends "I am alive" messages to the Administrative host. To ensure these periodic messages are sent the Basic datagram I/O package uses a timer/wake-up signal to initiate the transmission of the message to the Network Status Client. Because this is done by the I/O package and there is a copy of this package in every process that uses network I/O the network status data is collected on a per process not per host basis.

PVC Relay

A DNET relay used in the completion of DNET Permanent Virtual Circuits (PVCs).

Relay

Special DNET application processes located in a DNET gateway which perform protocol conversion for DNET streaming service between dissimilar networks. The appropriate Master Server process 'listens' on a particular protocol boundary when

idle and assigns a relay when a request for a protocol h'hop' is received from DNET.. The relays are named according to the protocol boundary which they are intended to bridge. Thus a T-D relay services requests which arrive on a TCP/IP network, relaying data to a DECnet net. Relays operate in a full duplex mode while actually in use.

Router

DNET employs a hierarchical routing strategy. Each DNET node has, for every (DNET) network known to it, information on the next DNET host to contact in order to move data toward the destination. The DNET router function uses the information in the routing table as follows: Given a destination network, host, and process, returns the next 'best' hop (network, host, process) to 'move' toward the destination.

Routing Table-

A hierarchical routing table that contains the next 'hop' from the local DNET host/network in the direction of all other DNET networks. A minimal version of this table is provided with the distribution copy of DNET. The table is currently maintained manually by the local system administrator. In the future, this table will be dynamically configured and maintained by the local DNET Network Status Server after intial startup has taken place. The routing table is named **tbls.net** and is located in the **dnet_home** directory.

Server Assignment Function-

Returns the name of an available server to a requesting Router, and updates the Domain Server Table.

Server Instance Table(s-

Lists the current specific instances of a particular class of DNET Application Server. Entries are made by the Master Server and cleared via dn_done() calls from the servers as they complete their tasks.

Server Registration Function-

This function is part of the Domain Server Process. It updates the Domain Server table with information from Servers (e.g."now in use").

DNET

PROGRAMMER' S REFERENCE

Version: 1.7
Print Date: 08/11/89 09:26:51
Module Name: prog.ref

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

NAME

dn_cdone - Free up user resources associated with a datagram communication endpoint.

SYNOPSIS

int dn_cdone()

DESCRIPTION

The dn_cdone library routine performs the cleanup of any resources allocated by dn_cinit(3U) and/or dn_chandler(3U).

Because the DNET Datagram Services are not implemented from the kernel, there is no feasible method for cleaning up after the user application unless explicately told to do so by the user application through the dn_cdone library routine. Because many of the datagram resources are stored in a shared user process, failure of the user applications to use the function will result in wasted space and resources to the point that no applications will work.

SEE ALSO

dn_cinit(3U), dn_chandler(3U)

RETURN VALUE

A value of 0 will be returned on success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_BADSTATE] The dn_cinit function was not called previous to invoking this function.

NAME

dn_chandler - Prepare for the asynchronous receipt of datagrams.

SYNOPSIS

```
#include "dnet.h"

int dn_chandler(dhandle, d_alert_sig, udg)
void (*dhandle)();
int d_alert_sig;
struct udg *udg;
```

DESCRIPTION

The dn_chandler library routine is used to provide a standard interface for the declaration of an exception handling routine for receiving datagrams asynchronously.

The address of your exception routine (a standard C function) is passed along with the address of a user datagram structure (udg). (Refer to the description of dn_cread(3U) for a description of the user datagram structure.) Upon the receipt of a datagram, the normal thread of activity of your program will be interrupted while the datagram is placed in the user datagram structure (The structure must be big enough). After successfully reading the datagram, the exception routine is called. The address of the user datagram structure is passed as the only argument. After returning from the exception routine, the normal thread of activity of your program is resumed.

In UNIX, the second argument is the signal number used to inform the library routines that a datagram is pending. The signal should not be used for any other purpose within your program. Although little validation is enforced upon the signal number chosen, it is suggested that either SIGUSR1, or SIGUSR2 is used. This signal number is ignored in VMS implementations.

While executing within or on behalf of your exception routine in UNIX environments, further indications of pending datagrams will be ignored. Before returning control to the normal thread of activity within your program, though, the library routines will ensure that no datagrams are pending. One signal then, may result in multiple invocations of the exception handling routine before control is returned to the normal thread of activity. The VMS environment provides for stacking of events which could have in similar results.

SEE ALSO

dn_cinit(3U), dn_cread(3U)

RETURN VALUE

A value of 0 will be returned on success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_AEXIST]	The process name that was requested to be bound to is already bound by a process in the state associated with Listen DGMS Service.
[D_BADARG]	The value of d_alert_sig was not in the range: (1-32)
[D_BADSTATE]	The calling process was not in a proper state to issue the dn_chandler function call. This error is identified by the dgms.
[D_BADNM]	The process name as bound to in the dn_cinit call was registered at the dgms as being null.

CAVEATS

The user datagram structure that you provide must be big enough to hold the biggest datagram that may arrive. The D_MAXDG constant may be used in combination with the dn_salloc library routine to create a structure large enough to hold the largest allowed datagram.

NAME

dn_cinit - Create a datagram communications endpoint.

SYNOPSIS

```
int dn_cinit(pname)
char *pname;    /* Optional specification of name to bind to */
```

DESCRIPTION

The dn_cinit library routine establishes a datagram communications endpoint over which datagrams may be received or sent.

If the endpoint is to be used for receipt of datagrams, a pname (process name) must be specified. This pname is the character string equivalent of the TCP/IP port number. A datagram is addressed via a network name, host name, and process name. The latter is used once on the proper machine to determine which dnet datagram server to send the datagram to. The dn_cinit routine will fail if the requested pname is already in use by another datagram service.

An empty string may be passed as an argument but will result in an endpoint not capable of receiving datagrams. The argument should, in all cases, point to a valid memory location to avoid an unrecoverable run-time error condition.

SEE ALSO

dn_cdone(3U)

RETURN VALUE

A value of 0 will be returned on success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_NODNET]	
[D_NOEXIST]	The above two errors are indication that the dgms process is not currently running.
[D_NOEXIST]	An internal error has occurred where the dgms process could not access this process.
[D_BADARG]	An internal error has occurred.
[D_AEXIST]	Another datagram service has already established an endpoint bound to the requested pname.
[D_NODGRSC]	The dgms is temporarily out of all allocated resources. This error may occur as a result of failure of datagram services to issue a dn_cdone successfully before ending.
[D_QUOTA]	Your quota limit has been exceeded. This should never occur with the current implementation since multiple datagram communications endpoints are not allowed.

NAME

dn_close - close a dnet communication channel

SYNOPSIS

int dn_close(chan)
int chan;

DESCRIPTION

The dn_close user library routine closes the dnet communication channel: **chan.**

SEE ALSO

dn_open(3U)

RETURN VALUE

A value of 0 will be returned on success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

NAME

dn_cread - read a datagram from a datagram communications endpoint.

SYNOPSIS

```
#include "dnet.h"

int dn_cread(dg, flag)
struct udg *dg;
int flag;
```

DESCRIPTION

The dn_cread library routine provides a method for reading datagrams synchronously, or within the normal thread of execution of your program. The **dg** argument should point to a user datagram structure large enough to hold the incoming datagram.

The default action of the dn_cread routine is to block until a datagram arrives, if an outstanding datagram does not exist. If this is not desirable, then the flag may contain the DG_F_NOWAIT bit set which will cause the dn_cread to return in error if no datagram is outstanding.

The following is a description of the user datagram structure:

```
struct udg
{
        struct node src;
        struct node next;
        struct node dest;
        long maxhops;   /* maximum number of hops before failure */
        int type; /* user defined type */
        long buflen;    /* length in bytes of buf */
        char buf[1];
};

struct node
{
        char host[I_MAXHNAME];
        char net[I_MAXNNAME];
        char proc[I_MAXPNAME];
};
```

The address of the user datagram is described in the **dest** node. The **src** and **next** nodes are set by the library routines. The next node is of transient significance to the datagram service itself. The src node may be examined by the server application to determine where the datagram came from. This field is stamped by the library routines on the way out and overwrites anything placed in it by the user routine.

All fields of the **dest** field should be filled in by the application before attempting to send the datagram. No methods exist for sending broadcast datagrams of any form.

The **maxhops** field is used to avoid errors in the routing tables which might cause a datagram to endlessly loop in attempt to get to it's destination node. This field is currently ignored as this service has not yet been provided.

The **type** field is currently not used by the system, although the range of types: (0-31) should be considered to be reserved types and should not be used. The type field is provided so that the user may have a standard mechanism for categorizing datagrams in whatever fashion needed.

The **buflen** field specifies the number of bytes of data in the **buf** field. The buf field is not limited to ASCII data, so special characters may be passed as part of the datagram.

The **buf** field hold the actual contents of the datagram. You may note that the buf field is defined as being one character long. The purpose of this is to allow the datagram applications to decide how long this field should be. This may be done by using the dn_salloc library routine to define an appropriately sized buffer and return an address which may be placed in a udg structure pointer variable.

SEE ALSO

dn_cinit(3U), dn_cwrite(3U), dn_chandler(3U), dn_cdone(3U)

RETURN VALUE

A value of -1 will indicate an error condition exists and the external variable dnet_errno can be checked to identify the error. A positive integer will represent the number of bytes contained in the message read.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_SHUTDOWN]	A shutdown message was received from the dgms. An attempt clean up will be attempted by the library routine and the datagram communications endpoint will be removed. The shutdown mechanism is not currently implemented and so this message should not be received.
[D_NOMSG]	The DG_F_NOWAIT flag was set and there were no outstanding datagrams.

CAVEATS

If the ipc mechanism used to communicate between the library routines and the dgms process fills up because of neglect, the dgms will begin discarding any newly received datagrams until there exists enough buffer space in the ipc mechanism to hold the entire datagram. The only indication of the datagram discarded as a result of this will be a terse error message in the dgms processes error output. This, though, is not the only possible cause for loss of datagrams in this unreliable datagram service.

The application must insure that the user datagram structure represents a buffer big enough to hold the largest datagram that might be received. The dn_salloc routine may be used with the DN_MAXDG constant to create the buffer necessary to hold the largest possible datagram.

NAME

dn_cwrite - Send a datagram to a remote process.

SYNOPSIS

```
#include "dnet.h"

int dn_cwrite(dg, flags)
struct udg *dg;
int flags;
```

DESCRIPTION

The dn_cwrite function call facilitates the sending of the datagram pointed to by **dg** to a remote process. Refer to the description of dn_cread(3U) for a discussion of the udg structure.

The **flags** argument does not currently have a use at the user level.

The datagram service is inherently unreliable. It is therefore the responsibility of the user processes to insure receipt.

SEE ALSO

dn_cread(3U)

RETURN VALUE

A value of 0 will be returned on success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_BADSTATE]	You have not successfully called dn_cinit yet.
[D_SHUTDOWN]	A shutdown indication has been sent by the dgms process.
[D_DGTB]	The **buflen** field was greater than the maximum allowable size of the entire datagram structure. The entire datagram structure must be less than D_MAXDG.

NAME

dn_done - connection services server completion routine

SYNOPSIS

int dn_done()

DESCRIPTION

This connection oriented user library routine should be called by all servers when they are finished servicing a particular client. An IPC mechanism is opened to the master server that tells the master server that this server is finished and ready for a new connection.

SEE ALSO

dn_getclient(3U)

RETURN VALUE

This routine returns the number of bytes written to the master server on success, and a value of -1 is returned to indicate that an error occurred.

ERRORS

The call fails if:

[D_INTERN] The call was unable to inform the dms module that it was available.

NAME

dn_getclient - Wait for a connect request from a remote client.

SYNOPSIS

int dn_getclient(service, usrbuf, pusrbuflen)
char *service;
char *usrbuf;
int *pusrbuflen

DESCRIPTION

The dn_getclient user library routine is called by a permanent server when it wants to establish a connection with a client which has requested its service.

The **service** argument points to a character string representing the name of your server. The **usrbuf** and **pusrbuflen** arguments describe a buffer in which the connection request message will be replaced which will contain the node identification of the requesting client.

If no requests are outstanding, this routine will block until a connection request arrives.

SEE ALSO

dn_done(3U), dn_close(3U)

RETURN VALUE

A positive value will be returned on success representing the channel descriptor back towards the requesting client. A value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

NAME

dn_init - initialize connection based dnet services

SYNOPSIS

#include "dnet.h"

int dn_init()

DESCRIPTION

This internal library routine is the dnet initialization function. It is called once after setting the progname, dlog and debug values. This function loads the nyname and network tables into memory.

RETURN VALUE

This routine returns a value of 0 on success and a value of -1 to indicate that an error occurred.

ERRORS

The call fails if:

[D_HOMELESS] The dnet_home environmental variable was not set.

CAVEATS

If dn_cinit is called from within your program, this routine should not be used.

NAME

dn_open - create a dnet communication channel

SYNOPSIS

int dn_open(destnet, desthost, destproc)
char *destnet;
char *desthost;
char *destproc;

DESCRIPTION

The dn_open user library routine establishes a dnet communication channel between the calling procedure and the specified server process on the specified host on the specified network. The server process will have previously issued the dn_getclient routine. The function does not return until the channel has been successfully established to the destination.

SEE ALSO

dn_write(3U), dn_read(3U), dn_close(3U)

RETURN VALUE

A positive value will be returned on success representing the channel number of the established communication channel. A value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

NAME

dn_login - verify username password for access to services on a node

SYNOPSIS

dn_login( )

DESCRIPTION

This library routine is used by DNET client processes which need to .

RETURN VALUE

This routine returns a value of 0 on success and a value of -1 to indicate that an error occurred.

ERRORS

The call fails if:

CAVEATS

NAME

dn_login_verify - verify username password for access to services on a node

SYNOPSIS

dn_login_verify( )

DESCRIPTION

This library routine is used by DNET server processes which need to verify that the current user has access privileges on the local DNET host.

RETURN VALUE

This routine returns a value of 0 on success and a value of -1 to indicate that an error occurred.

ERRORS

The call fails if:

CAVEATS

NAME

dn_read - read data from a dnet communication channel

SYNOPSIS

```
int dn_read(channel, buf, nbytes)
int channel; /* pointer to channel created with dn_open */
char *buf;
int nbytes; /* Maximum number of bytes to read */
```

DESCRIPTION

The dn_read user library routine allows data to be read from a channel created previously with the dn_getclient(3U) or dn_open(3U) library routines.

SEE ALSO

dn_write(3U)

RETURN VALUE

A positive value representing the number of bytes read will be returned on success. A value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

NAME

dn_write - write data on a dnet communication channel

SYNOPSIS

int dn_write(channel, bu ', nbytes)
int channel;
char *buf;
int nbytes;

DESCRIPTION

The dn_write user library routine writes **nbytes** bytes from **buf** onto the channel: **channel.**

SEE ALSO

dn_open(3U), dn_getclient(3U), dn_read(3U)

RETURN VALUE

A value of 0 will indicate success, and a value of -1 will indicate an error.

ERRORS

The call fails if:

[D_SGXXX

DNET

ADMINISTRATOR' S GUIDE

Version: 1.31
Print Date: 09/28/89 10:58:01
Module Name: admin.gui

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

CONTENTS

1. DNET Administration Overview 1
   1.1 Introduction 1
   1.2 Overall Network Concerns 1
   1.3 Local Host Administration 1

2. Distribution of DNET Software 2
   2.1 Modification of target machine database - tbls.db 2
   2.2 Creating the distribution files 3
   2.3 Moving DNET Source Files to Target Machine 3
   2.4 Generating the target files 3
      2.4.1 UNIX Machines 3
      2.4.2 VAX Machines 4
   2.5 Use of ptar to pack/unpack files 4
   2.6 Making the Target Executables for DNET on the local machine 4
      2.6.1 UNIX 4
      2.6.2 VMS 4

3. Initial Configuration of Local (non-gateway) DNET Node 6
   3.1 Environment & Special Permissions 6
      3.1.1 General 6
      3.1.2 UNIX 6
      3.1.3 VMS 7
   3.2 DNET Tables & Local Host 'Service' Files 11
      3.2.1 Services Files 11
      3.2.2 tbls.myname - Local Host Name(s) file 12
   3.3 Adding/Deleting/Modifying Servers at a DNET host 12
      3.3.1 Types of Servers 12
      3.3.2 Control of Servers 12
      3.3.3 Number and Types of Servers 12
      3.3.4 Prespawning of Servers 13
      3.3.5 Maximum Number of Servers 14
      3.3.6 Adding/Removing Servers 14
   3.4 Datagram Service Administration 14
      3.4.1 Normal Operation 14
      3.4.2 The Static Backbone Network 14
   3.5 DNET Routing 14
      3.5.1 Router Operation 14
      3.5.2 Routing Example 15
      3.5.3 Routing Table Updates 16
      3.5.4 Future Enhancement of Router Operation 16

4. Gateway Administration 17
   4.1 PVC Relays 17
   4.2 Relay of Datagrams 18

5. DNET Start-up on an Individual DNET Host 19
   5.1 UNIX 19
      5.1.1 Individual Scripts 19
   5.2 VAX VMS 19
      5.2.1 Individual Scripts 19

6. DNET Shutdown 21
   6.1 UNIX 21
   6.2 VAX - VMS 21

7. Network Startup 22

8. Network Administration Operations 23
8.1 Network Maintenance 23
8.1.1 Adding an additional DNET Host Site 23
8.1.2 Deactivating an existing DNET Host Site 23
8.1.3 Adding an additional DNET Network 23
8.1.4 Deactivating an existing DNET Network 23

9. Testing a DNET Installation 24

10. DNET Initial Demonstration Network 25
10.1 Network Topology 25
10.2 Information on DNET nodes 27
10.3 Starting up (a subset of) the Demonstration Network 27

11. Asynchronous DECnet connection from d a c v a x to SPANET 29
11.1 Starting the Link 29
11.2 Stopping the asynch DECnet link 29

12. DNET Network Utility Commands 31
12.1 Examining The Status of DNET 31
12.2 Testing if DNET is alive 31
12.3 Obtaining Status of DNET Servers 32
12.4 Underlying Processes for Network Status 34
12.4.1 Update Local Routing Table 34

13. DNET Errors 35

14. DNET Security 37
14.1 Execution Security 37
14.2 User Security 37
14.2.1 UNIX 37
14.2.2 VMS 37
14.3 File Security 37

15. Electronic Mail Administration 38

16. Library and Program Pool Administration 39

17. DNET Performance Monitoring 40
17.1 General 40
17.2 DNET Performance Test Application - dptc 40
17.3 VMS Host vs UNIX Host 40
17.4 DECnet vs TCP/IP 40

18. Glossary 41

1. DNET Administration Overview

1.1 Introduction

Administration of DNET is divided into two general categories. These categories relate to 1) overall DNET network issues and 2) local DNET node administration.

1.2 Overall Network Concerns

The following are the major 'global' concerns in the administration of DNET.

1. Admininistration of Underlying Networks

   Since DNET operates as a meta-network or a network of networks, its operation is highly dependent on the intergrity of the underlying networks such as TCP/IP and DECnet. These networks are maintained in their ordinary fashion; under normal circumstances so long as the underlying network(s) are operational it should be possible for DNET to use these network(s) for its purposes. The behavior of DNET may be affected by any or all of the following factors:

   1. Changes in Local Operating System
   2. Upgrades or changes in local network interfaces

2. DNET Network Map

   The master 'copy' of the DNET network map is maintained at -- TO BE DETERMINED This factor influences, at a minimum, the contents of DNET routing tables.

3. Consistency of Underlying Network Names in DNET Tables

4. Routing Strategies in DNET

   Routing tables are currently 'static'. They are loaded when the network is started and are not updated while the network is operating. DNET mechanisms could be used to may these tables dynamic in the future.

1.3 Local Host Administration

Once DNET software has been installed on a particular computer, administration of the local DNET node generally involves the specification of the number and types of DNET application servers which will be allowed to operate on the local node, adjustment of quotas and permissions as necessary, and administration of the DNET routing tables and mail.

2. Distribution of DNET Software

2.1 Modification of target machine database - tbls.db

If the target machine is 'new' to DNET, it is convenient to add it to the DNET environment 'data base' prior to making a distribution copy. This is done by editing the 'master' copy of the file **tbls.db** in the sccs_src directory on 'stubby' as follows:

1. cd $dnet_home
2. cd ../sccs_src
3. get -e s.tbls.db
4. locate a convenient entry which is similar to the target machine

   NOTE: Alternatively, one of the default machine entries, **bsd_dflt, sysv_dflt,mvax_dflt, or vax_dflt** may be used during the **postmove** operation described below to obtain a default configuration.

   An examples for a UNIX host is shown below:

```
                    IUESN1 - NASA GSFC, Greenbelt, MD
iuesn1|envname|DNSUN4B
iuesn1|myname|#mynet myhost
iuesn1|myname|starnet iuesn1
iuesn1|net|#destnet nexthost relay    nextprotocol
iuesn1|net|starnet      NULL  NULL           tcp
iuesn1|net|dnett1       dftnic drelaytd tcp
iuesn1|net|spanet       dftnic drelaytd tcp
iuesn1|msinittcp|dechod       dechod       1     1     1
iuesn1|msinittcp|drexecd      drexecd      1     1     1
iuesn1|msinittcp|dtftpd       dtftpd       1     1     1
iuesn1|msinittcp|dlogind      dlogind      1     1     1
iuesn1|msinittcp|dmaild       dmaild       0     1     1
iuesn1|msinittcp|dncld        dncld        0     3     3
```

   An example entry for a VAX which is also a DNET gateway machine is shown below:

```
                DFTNIC          - NASA GSFC, Greenbelt, MD
dftnic|envname|DNDFTNIC
dftnic|myname|#mynet  myhost
dftnic|myname|spanet  dftnic
dftnic|myname|starnet dftnic
dftnic|net|#destnet       nexthost relay    nextprotocol
dftnic|net|spanet         NULL     NULL           dec
dftnic|net|dnett1         dacvax   drelaydt dec
dftnic|net|starnet        NULL     NULL           tcp
dftnic|msinitdec|drelaydt         drelaydt1      1       1
dftnic|msinitdec|dechod           dechod         1       2       2
dftnic|msinitdec|drexecd          drexecd        1       1       1
dftnic|msinitdec|dtftpd           dtftpd         1       1       1
dftnic|msinitdec|dlogindd logind        0        1       1
dftnic|msinitdec|dmaild           dmaild         0       1       1
dftnic|msinitdec|dncld            dncld          0       3       3
dftnic|msinittcp|drelaytd         drelaytd       1       1       1
```

5. copy this entry to the bottom of the file and change the machine name to that of the target in all pertinent locations.

2.2 Creating the distribution files

The archival SCCS copy of the DNET software is found on the master DNET host, currently **dac3b2**, an AT&T 3B2-600 located at DAC. A master copy of the DNET software may be obtained at any time from this machine and placed in a form for distribution to any target machine. The steps to generate the distribution copy are as follows:

1. login to **stubby** or **dac3b2**
2. cd /mnt/comm/dnet/bin - stubby

   /usr/nasa/dnet/bin - dac3b2

   makemove

At the completion of the **makemove** operation, the directory /tmp/dnet_move will contain the following 'ptar' & other files:

dnet.ptar
pvc.ptar
app.ptar
common.ptar
dg.ptar
ptar.ptar
postmove
postmove.vms
dman

Further details on **makemove** are provided in the **DNET Administrative Reference Manual.**

2.3 Moving DNET Source Files to Target Machine

The files generated by 'makemove' and placed in **/tmp/dnet_move** should be moved to the target machine using **FTP** and/or **copy** depending on the network(s) involved.

The target directory for these files will differ depending on the target machine:

PTAR Directory

1. UNIX hosts - **/tmp/dnet_move**
2. VAX hosts - **dnet_home:[.dnet]** where dnet_home is an arbitrary path

2.4 Generating the target files

2.4.1 UNIX Machines

1. Transfer the 'ptar' files,'postmove', and postmove.vms to the target machine
2. cd **dnet_home**/bin
3. postmove -hnXXX **dnet_home**

   where

   XXX is name of this local host &
   (or a default name chosen from bsd_dflt, sysv_dflt, mvax_dflt, or vax_dflt)

   dnet_home is an arbitrary path

2.4.2 VAX Machines

The procedure for VAX machines differs only slightly from that on UNIX hosts. The following steps should be performed:

1. Transfer the 'ptar' files to the target machine and place the files in the **dnet_home** directory
2. login to the target machine
3. cd dnet_home
4. @postmove.vms
5. Enter the name of the local machine when prompted
   (or a default name chosen from bsd_dflt, sysv_dflt, mvax_dflt, or vax_dflt)
6. Wait for postmove to complete unpacking and distributing the files

2.5 Use of ptar to pack/unpack files

The **ptar** program allows the packing/unpacking of files in a generic format for transfer to DNET target machines. Ordinarily, postmove automatically extracts files from the ptar files, however this extraction may be performed manually, if necessary.

ptar -x file.ptar

2.6 Making the Target Executables for DNET on the local machine

There are a number of 'make' files which are included with the DNET distribution package. The **postmove** operation automatically places these files in the appropriate directories on the target machine and updates the necessary environment variables within the files to the target computer. Thus, typically one need only start the 'make' procedures in order to generate a current copy of the DNET executable files. The specific procedures are outlined in the following sections.

2.6.1 *UNIX*

1. If this is a first time installation on this UNIX host, follow steps above for setting environment variables, etc.
2. cd **$dnet_home**
3. make
4. wait for the make process to complete

2.6.2 *VMS*

1. If this is a first time installation on this VAX, follow steps above for setting environment variables, etc.
2. cd **dnet_home**
3. The exact **make** file used will depend on the VAX environment as follows:
    1. dacvax (or other MicroVAX with VMS and Wollongong TCP/IP Interface)

        make.dv

    2. NASA Vaxes with Excelan Interface (or other VAX with VMS and Excelan TCP/IP Interface)

        make.dft
4. Wait for the make procedure to complete

3. Initial Configuration of Local (non-gateway) DNET Node

This section describes how to configure a Local DNET node which is not a gateway node. Special considerations for gateway nodes are described in a later section.

3.1 Environment & Special Permissions

3.1.1 General

Environment Variables/Logical Names

The following 'environment' variables are used by all DNET software.

1. dnet_home - the 'home' directory of the DNET software
2. dnet_gateway = 1 if machine is a DNET gateway
3. path to dnet_bin - the directory containing the DNET executables
4. dnet_debug - this flag controls the generation of various debugging 'log' files; it should ordinarily be set to 0. It should be set to 1 if the debug 'log' option is desired (see DNET PROGRAMMER's GUIDE)

While these general requirements apply to both the UNIX and VMS environments, the specific details differ considerably between the two operating systems. The specifics are covered in the following sections.

3.1.2 UNIX

The environment variables may be set in UNIX by modification of the user **.profile** file found in each users home directory.

Additions to **.profile** for DNET:

Bourne shell

```
dnet_home = /..../ ... /dnet; export dnet_home

PATH=existing path specs;/dnet_home/bin

. $dnet_home/dnlogin.sh
```

C shell

```
setenv dnet_home /..../ ... /dnet

PATH=existing path specs;/dnet_home/bin

source $dnet_home/dnlogin.csh
```

3.1.3 VMS

Specification of the DNET 'environment' is somewhat more complex for VAX/VMS systems. The correct operation of DNET requires that certain VMS **Privileges** and **Quotas** be set in addition to the usual environment variables.

3.1.3.1 General Enviroment Variables - Logical Names

The general DNET environment variables are set in VMS using The **login.com** file in the VAX login directory should contain the following lines. The entries define logical names in the 'GROUP' table.

The values are for the **dacvax** machine

```
$ define/group dnet_home "$disk1:[sys0.dnet.dnet]"
$ define/group dnet_bin "$disk1:[sys0.dnet.dnet.bin]"
$ define/group dnet_gateway 1
```

For **IAF and DFTNIC** these definitions should be:

```
$ define/group dnet_home "cldata:[dnet.dnet]"
$ define/group dnet_bin "cldata:[dnet.dnet.bin]"
$ define/group dnet_gateway 1
```

If the dnet_debug option is desired, it should be set in a 'transient' fashion in the 'JOB' table as follows:

```
$ define/job dnet_debug 1
```

Ordinarily, most of the VMS environment can be set 'automatically' using script files provided with the distribution. These scripts are executed as part of the usual 'login' procedure. Only a short 'machine specific' change should usually be required in the **login.com** file. This change is accomplished as follows:

1. cd sys$login
2. Edit the file **login.com** to add the following entries:

```
$
$! DNET Specific Environment
$ set proc/priv=grpnam
$ define/group dnet_home $disk1:[sys0.dnet.dnet]
$! run dnetlogin script
$ @dnet_home:dnlogin.dv
$
```

   NOTE: The specifications for **dnet_home** & **dnlogin** are machine specific. The example given for **dnet_home** for the DAC MicrovaxII, **dacvax**.

3. The corresponding version of **dnlogin.xx** is **dnlogin.dv** which is located in the dnet_home directory. The contents of **dnlogin.dv** are shown below: All dnlogin.xx files are included with the DNET source code listings.

```
$! dnlogin.com
$! login script for DNET
$
$! logical names
$
$ DNET_DEBUG = = "0"
$ define wool_netinet $disk1:[net.wool.netdist.include.netinet]
$ define wool_sys $disk1:[net.wool.netdist.include.sys]
$
$ define/job dnet_pvcdir $disk1:[sys0.dnet.dnet.pvcdir]
$ define/job dnet_dgdir $disk1:[sys0.dnet.dnet.dgdir]
$ define/job dnet_common $disk1:[sys0.dnet.dnet.common]
$ define/job dnet_appdir $disk1:[sys0.dnet.dnet.appdir]
$! define/job dnet_debug 1
$ set proc/priv=grpnam
$ define/group dnet_home $disk1:[sys0.dnet.dnet]
$ define/group dnet_mail $disk1:[sys0.dnet.dnet.mail]
$ define/group dnet_bin $disk1:[sys0.dnet.dnet.bin]
$ define/group dnet_gateway 1
$
$ define c$include dnet_common, dnet_pvcdir, dnet_dgdir, wool_sys, -
 wool_netinet
$ define vaxc$include c$include, sys$library
$
$ assign $disk1:[user.net.libnet.dacnet] TABLES
$
$! Clients
$
$ decho  = = "$ dnet_bin:decho.exe"
$ ddechoc  = = "$ dnet_bin:ddechoc.exe"
$ dechon  = = "$ dnet_bin:dechon.exe"
$ drexec  = = "$ dnet_bin:drexec.exe"
$ dtftp  = = "$ dnet_bin:dtftp.exe"
$ dlogin = = "$ dnet_bin:dlogin.exe"
$ dmskill = = "$ dnet_bin:dmskill.exe"
$ dnetstat = = "$ dnet_bin:dnetstat.exe"
$ dncl = = "$ dnet_bin:dncl.exe"
$ dmail = = "$ dnet_bin:dmail.exe"
```

```
$
$! development only
$
$! delmbx   = = "$ $disk1:[odnet]delmbx.exe"
$ shack = = "$ dnet_bin:shack.exe"
$ i_to_o = = "$ dnet_bin:i_to_o.exe"
$ bcd = = "$ dnet_bin:bcd.exe"
$ ddechoc = = "$ dnet_bin:ddechoc.exe"
$
$! aliases
$
$ sl = = "show logical"
$ ss = = "show symbol"
$ ls = = "dir"
$ l = = "dir"
$ cd = = "set def"
$ pwd = = "show def"
$ vi = = "ed"
$ view = = "ed/read_only"
$ ps = = "show system"
$ ns = = "netstat -a"
$ more = = "type/page"
$ clear = = "@clear"
$
$ ll = = "$ $disk1:[user.net.libnet.paul]ll.exe"
$ ptar = = "$ $disk1:[sys0.dnet.bin]ptar.exe"
$ od = = "$ $disk1:[user.net.libnet.paul]od.exe"
$ wc = = "$ $disk1:[user.net.libnet.paul]wc.exe"
$
$ set proc/priv=sysnam
$
$ cd dnet_home
$ checkdmail
$
```

3.1.3.2 Privileges

VMS has an extensive set of privileges which control the various operations which a user or process may perform on the VAX. The following privileges are required for DNET operation.

1. SYSNAM

   This privilege is required for DECnet network operations; DNET servers will not operate without this privilege

2. GRPNAM

   This allows logical names to be placed in the 'GROUP' table; This table is a convenient location for the DNET environment variables.

3. GROUP

4. NETMBX

Allows creation of 'network' mailboxes

5. TMPMBX

   Allows creation of 'temporary' mailboxes

These privileges need to be 'activated' in order to be used. The **dnlogin.xxx** file in the VAX login directory should contain the following lines:

$ set proc/priv=grpnam
$ set proc/priv=sysnam

3.1.3.3 Quotas

VMS defines a large number of resource controls known as 'quotas'. Certain of these quotas must be set to other than their default values in order to successfully operate DNET. An annotated list of the pertinent quotas is given below. The following section describes how to change these quotas.

1. Byte Count Limit BYTLM - 60000

   This quota determines the temporary storage available to DNET for mailboxes. Each active DNET process (including Master Servers and application servers) requires a minimum of 2 mailboxes for its operation.

   Approximate Formula for Determing appropriate Byte Count Limit

   #of Entries in tbls.msinit (dec & tcp) = AS

   2 * (DGMS + DMS + AS) * 2000

2. Job Table Quota

   JT Quota - 12000

   This value controls the amount of information which DNET can place in the JOB Table

3. Paging File Quota - 30,000

   This quota is used as an adjunct to swap operations in VMS and needs to be increased as the number of DNET processes increased.

4. AST

   Controls the number of simultaneous AST operations allowed by DNET; this value will probably need to be increased in gateways where a large number of DNET relay processes are in use.

5. Subprocess Quota - PRCLM - 30

   This Quota controls the number of subprocesses which may operate under DNET. The exact number will undoubtedly be controlled by the local system adminstrator. Value should match that of MAXDETACH.

6. Open File Limit - FILLM - 300

7. Max Detached Processes - MAXDETACH - 30

   This value controls the number of detached processes which can be started by DNET. This quota is important when DNET is started in 'stand-alone' or detached mode. Value should match that of PRCLM.

3.1.3.4 Setting/Changing Quotas

The procedure for changing quotas on the VAX is as follows;

1. Login as 'SYSTEM'
2. cd sys$system
3. Run Authorize
4. UAF>

   modify dnet /prclm=30000

   etc.

The command **show/full dnet** may be used to list all of the quotas while in 'AUTHORIZE'

3.2 DNET Tables & Local Host 'Service' Files

Several Files (& Tables) must be modified for the local DNET node.

1. Network 'Services' File
2. Initial Local Version of DNET Routing Table - **tbls.net**
3. User Alias Table - **tbls.myname**
4. Master Server Init Table - **tbls.msinittcp** and/or **tbls.msinitdec**
5. Connection Lock Table (for Datagram Backbone Network - not yet implemented)

Modification of each of these files is discussed below:

3.2.1 Services Files

Service Files on the Local Machine must be modified to support DNET.

1. UNIX

   The standard file

   /etc/services

   must contain the following entries:

```
5279    dms/tcp       # DNET PVC Master Server
5279    dgsudp/udp    # DNET UDP Datagram Server
```

These entries can only be changed by 'root' and need be made only when DNET is first installed on the UNIX machine.

2. VAX/VMS - No special changes are required to register DNET servers on DECnet.

3.2.2 tbls.myname - Local Host Name(s) file

The file **tbls.myname** found in the **dnet_home** directory contains one or more names for the local host. This file allows "self-identification" of the local host by DNET software and is used by the routing function.

An example of the 'tbls.myname' file is shown below:

DNET Local 'myname' Table	
Name	**Network**
dacvax	**spanet**
dacvax	**dnett1**

3.3 Adding/Deleting/Modifying Servers at a DNET host

3.3.1 Types of Servers

There are two application server types defined within DNET:

1. **DNET Application Servers** - called by client processes, these service providers include a DNET Basic I/O package. For all these services (File Transfer, Network Command Server, other application servers) there is a process which spawns copies of them and assigns the copies to clients on request. This controlling process is the "DNET Master Server".
2. **Other Servers (user defined, etc.)** - spawned via DNET network command server (net_com_serv) these servers do not contain the DNET Basic I/O Package. They depend on the network command server to interface with DNET.

3.3.2 Control of Servers

The control of DNET servers which require streaming service is under the control of the **DNET Master Server** at each DNET host. These servers may be either prespawned or spawned on demand depending on the type of host and local system considerations.

Bidirectional connectionless service is also available to these servers if they register with the Datagram Master Server. Details of connectionless operations are provided in a later section.

3.3.3 Number and Types of Servers

The system administrator on a particular DNET host controls the number and types of DNET servers which operate on that host.

The number and types of servers are determined by the DNET Master Server Table Init file:

This is a 'flat' ASCII file. Entries in the file appear on separate rows and have the format as follows:

DNET Master Server Init Table				
Server Type	Image Name	# Prespawned	Max #	Init #
dechod	dechod	1	8	3
dtftpd	dtftpd	1	9	4
dremc	dremc	1	1	1
dnstatd	dnstatd	1	1	1
dncld	dncld1	1	10	5
dlogind	dlogind	1	10	5
dmaild	dmaild	1	10	1

The number of prespawned servers is specified in column 3.

The Maximum (permissible) number of servers of this type is specified in column 4

Column 5 contains the number of servers to be started when DNET is first started

Servers may be added or deleted by editing this file (DNET admin privileges required)

Further discussion of the significance of these entries is provided in the following sections.

A separate Master Server Init File is required for each protocol connection at a DNET host. Thus, at a VAX which is connected to both a TCP/IP and a DECnet Network, there must be two such tables **tbls.msinittcp** and **tbls.msinitdec**.

3.3.4 Prespawning of Servers

In order to improve the efficiency of response for DNET service requests on VAX machines, certain DNET servers may be 'prespawned' prior to service requests.

The number and type of prespawned servers is specified in the Master Server Init Table File described in the preceding section.

Possible algorithms for spawning and assignment are:

1. At network start up, spawn a number of copies of the servers, according to the contents of the **DNET Master Server Init Table** keeping their process id's for later use in forming the process names to give to clients. After allocating a server to a client, spawn another to replace it.
2. For less frequently used services- Spawn only when a client requests a server. This is the Transient Server.
3. For very frequently used services- Spawn the maximum number desired and have servers listen for the next client when they complete their service for a client, and at the same time notify the Master that they are ready for assignment.

3.3.5 Maximum Number of Servers

This parameter controls the maximum number of simultaneous copies of a particular server which are allowed at the local host. This number can be adjusted by the system administrator according to conditions on the local system.

3.3.6 Adding/Removing Servers

The following steps are used to control DNET application servers.

1. Edit the Master Server Init Table (**tbls.msinittcp** &/or **tbls.msinitdec**) found in the directory **dnet_home**
2. Scroll to desired row of table and type in the new entry according to the format described below.
3. Write the table back, overwriting the existing table.
4. The new version of the Master Server Init Table will be read automatically when the Master Server is started.

3.4 Datagram Service Administration

3.4.1 Normal Operation

Under normal circumstances, the datagram service requires no action on the part of the system administrator.

3.4.2 The Static Backbone Network

This feature is currently not implemented in DNET.

3.4.2.1 Adding Elements

3.4.2.2 Removing Elements

3.5 DNET Routing

This section describes the operation and control of routing within DNET from the perspective of the local system administrator.

3.5.1 Router Operation

The paths to hosts in the local network are direct connections via usual local network mechanisms. For paths to hosts in other networks a dynamic router is used. A hierarchical routing table is used to determine the next host to which a PVC connection request or a connectionless datagram should be forwarded to 'move' toward the final destination.

A typical routing table is shown below:

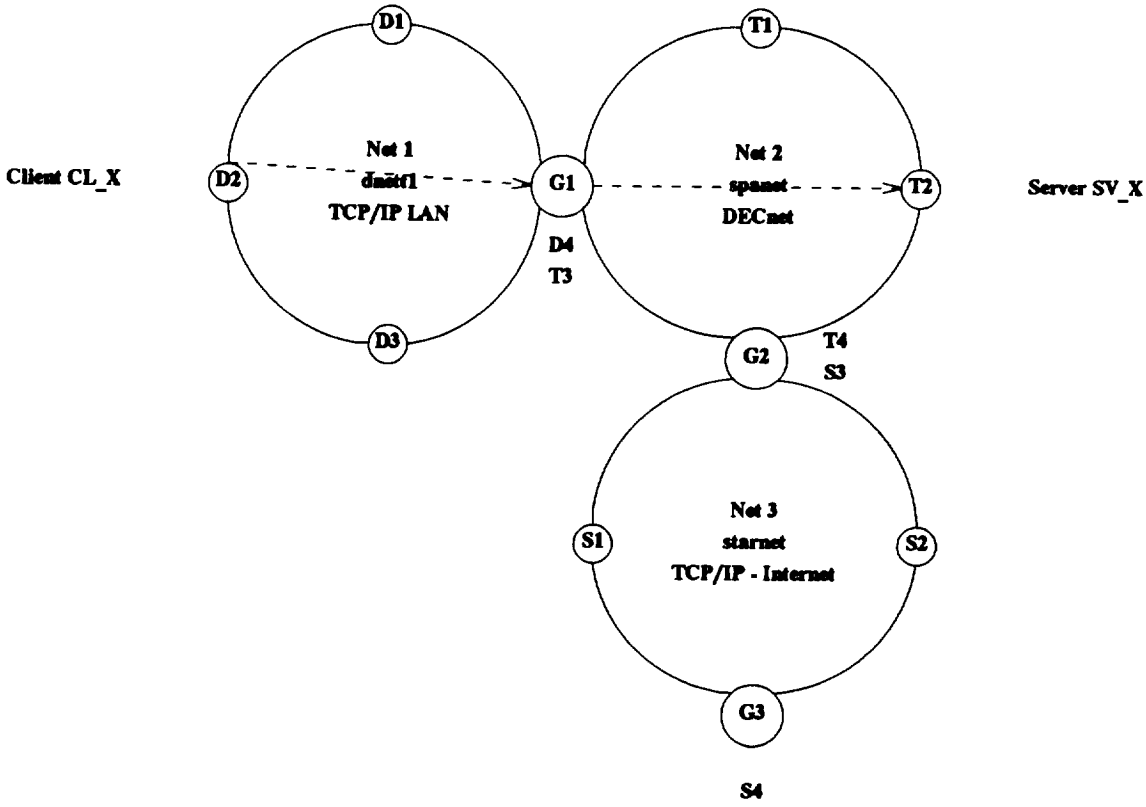
DNET Local Routing Table			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
dnett1	-	-	udp
spanet	dacvax	drelaytd	udp
starnet	dacvax	drelaytd	udp
Net X	Host Y	drelaytX	udp
-	-	-	-

The four columns in the routing table contain the following information

1. Destination Network
2. Next Host (in hierarchical path to destination net)
3. Next DNET Process (always a relay except for last hop)
4. Local Datagram protocol used to make next hop

3.5.2 Routing Example

The route generated for a typical datagram is shown in the following diagram:



In this example client CL_X on DNET host D2 wishes to conduct a session with server SV_X on DNET host T2.

The router on host D2 has the following routing table available:

DNET Local Routing Table - Host D2			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
dnett1	NULL	NULL	udp
spanet	dacvax	drelaytd	udp
starnet	dacvax	drelaytd	udp
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

The router on host D4 has the following routing table available:

DNET Local Routing Table - (Gateway) Host D4			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
spanet	NULL	NULL	dec
dnett1	dacvax	drelaytd	udp
starnet	inf	delaydt	dec
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

3.5.3 Routing Table Updates

Initially, routing table updates will be handled in a manual fashion. Examination of a method for automatic updates for these tables will is a topic for further expansion of DNET as discussed in the next section.

3.5.4 Future Enhancement of Router Operation

In the future the router may be enhanced to include searching for alternate paths and servers if the standard search fails to satisfy the request. The second search could extend into other networks in requests for generic servers that need not be executed in a specific network or host. Extended searches will provide automatic alternate routing, load sharing, and backup services for use when failures in hardware or software reduce the availability of facilities. The entries in the routing table are updated by exchange of connectionless datagrams between DNET gateways and individual DNET hosts.

4. Gateway Administration

DNET gateways are similar to ordinary DNET hosts but, in addition, they have connections to at least two underlying network (protocols) supported by DNET. There is a PVC Master Server and a pair of per-protocol datagram servers for each of these protocols and an a pre-specified number of inter-protocol PVC 'relay' processes. The exact number of the latter is indicated in the appropriate Master Server Init Table.

4.1 PVC Relays

These relay processes are named according to the Master server with which they are associated and for the protocol pair for which they provide conversion service. The general naming convention is

drelayXY_n where

X is a single letter representing the protocol of the associated master server

Y is a letter representing the protocol to which conversion must be made.

n is the nth instance of this relay; used to provide a unique name for the relay server

Thus for a typical DNET TCP/IP - DECnet gateway machine the 1st instance of a relay associated with the TCP/IP master server (dmstcp) and providing conversion to DECnet is named:

drelaytd_1

Similarly the 3rd instance of a relay associated with the DECnet master server (dmsdec) and converting to the TCP/IP protocol is named:

drelaydt_3

The PVC Master Server Init Table for a Typical Gateway is shown in the following diagram:

DNET Master Server Init Table - Typical Gateway Machine				
Server Type	Image Name	# Prespawned	Max #	Init #
dechod	dechod	1	8	3
dtftpd	dtftpd	1	1	1
dremsc	dremsc	1	1	1
dnstatd	dnstatd	1	1	1
dncld	dncld1	1	10	2
dlogind	dlogind	1	3	1
dmaild	dmaild	1	10	1
drelaydt	drelaydt	1	10	5
drelaytd	drelaytd	1	10	5

4.2 Relay of Datagrams

Routing of datagrams is accomplished by the Datagram Master Server at each DNET node. The routing information for datagrams is included as the last

5. DNET Start-up on an Individual DNET Host

Three Administrative 'Script' Programs are used to control DNET on a local host machine:

1. dnstart
2. dnstop
3. dnadmin

The sequence of operations necessary to 'start' DNET on a local DNET host is given below. The steps vary in their details according to the type of machine/operating system.

5.1 UNIX

1. DNET Software is loaded onto two or more hosts and at least one DNET gateway.
2. The local Master Server Init Table, Host Alias Table, DNET Routing Table are checked for accuracy and edited as necessary
3. The command script

   dnstart

   is invoked in order to start the necessary processes on the local host. (NOTE: This script is added to the reboot procedure for the DNET host machines.) Once this script is invoked, the DNET Master Server Process and the protocol specific DNET datagram servers becomes operational as a DNET Well Known Servers.
4. Each DNET Master Server then spawns (or initializes in anticipation of spawning) the servers indicated in its Master Server Init Table. This produces the initial set of servers (File Transfer, Remote Login, Command Language Processor, etc).

5.1.1 Individual Scripts

dnstart invokes three other scripts to start the several DNET components. The components and associated scripts are:

1. Datagram Service Script - **strdgms**
2. PVC Service Script - **strpvc**
3. Network Status Service Script - **strstat**

5.2 VAX VMS

1. cd dnet_home
2. @dnstart

5.2.1 Individual Scripts

1. Datagram Service Script - **strdgms.com**
2. PVC Service Script - **strpvc.com**
3. Network Status Service Script - **strstat.com**

6. DNET Shutdown

DNET shutdown is done by the following:

1. The Administrative Server sends a Datagram to each Domain Server requesting that they shutdown all activity. This means no further service requests will be processed and all active server processes, as indicated in the Domain Server Table, will be sent "ABORT" signals.

   The Administrative Server may then be terminated, or left in an idle state until the next network start-up.

NOTE: If the local node is a DNET Gateway, shutdown may adversely affect the operation of DNET.

6.1 UNIX

1. cd $dnet_home
2. cd bin
3. dnstop

6.2 VAX - VMS

1. cd dnet_home
2. @dnstop
3. Wait for "stopping ...." messages to complete

7. Network Startup

There is no global activation procedure for DNET. Since DNET is a meta-network, the integrity of the DNET network is dependent on the following.

1. Required Hosts are Operational
2. Underlying Networks are Operational
3. DNET Processes Operating at all nodes required to reach a particular destination - i.e. local administrators must have activated DNET at each of these nodes

If these conditions are met, DNET should operate, within the limitations of loading on each of the nodes.

The **dnetstat** function may be used to examine the integrity of the network, if required (see section below).

8. Network Administration Operations

The following administrative operations are possible for the network as a whole.

- Modify DNET configuration
- Add/Delete Underlying Networks
- Add/Delete Local Hosts
- Examine and Modify Administrative Tables

8.1 Network Maintenance

8.1.1 Adding an additional DNET Host Site

This is a local operation.

8.1.2 Deactivating an existing DNET Host Site

This is a local operation.

8.1.3 Adding an additional DNET Network

1. One or more hosts in new network must have DNET software installed and operational
2. DNET Gateway(s) into the new network must be identified, have DNET software installed, and be operational
3. Routing Tables must be updated to include new destination network and appropriate gateway(s)

8.1.4 Deactivating an existing DNET Network

If a network is to be removed from DNET, this can be accomplished by deleting this network from the routing tables.

9. Testing a DNET Installation

This section describes the functional testing of DNET operation at a local node.

1. If you have not done so already, edit the appropriate Master Server Init Tables for this node. Make sure that at least one echo server, **dechod**, is specified for this node.
2. Start DNET on the local node following the procedure described in an earlier section.
3. Run **decho** and attempt to contact the local node following the instructions in the DNET User's Guide. If you are able to run the echo program, the PVC service is probably ok at this node
4. At the shell prompt, type **dnetstat**. If a short form list of the DNET servers on this node is printed, most other local DNET functions are probably working normally.
5. If another node on the local net is operational, try using dnetstat to 'ping' this node by entering

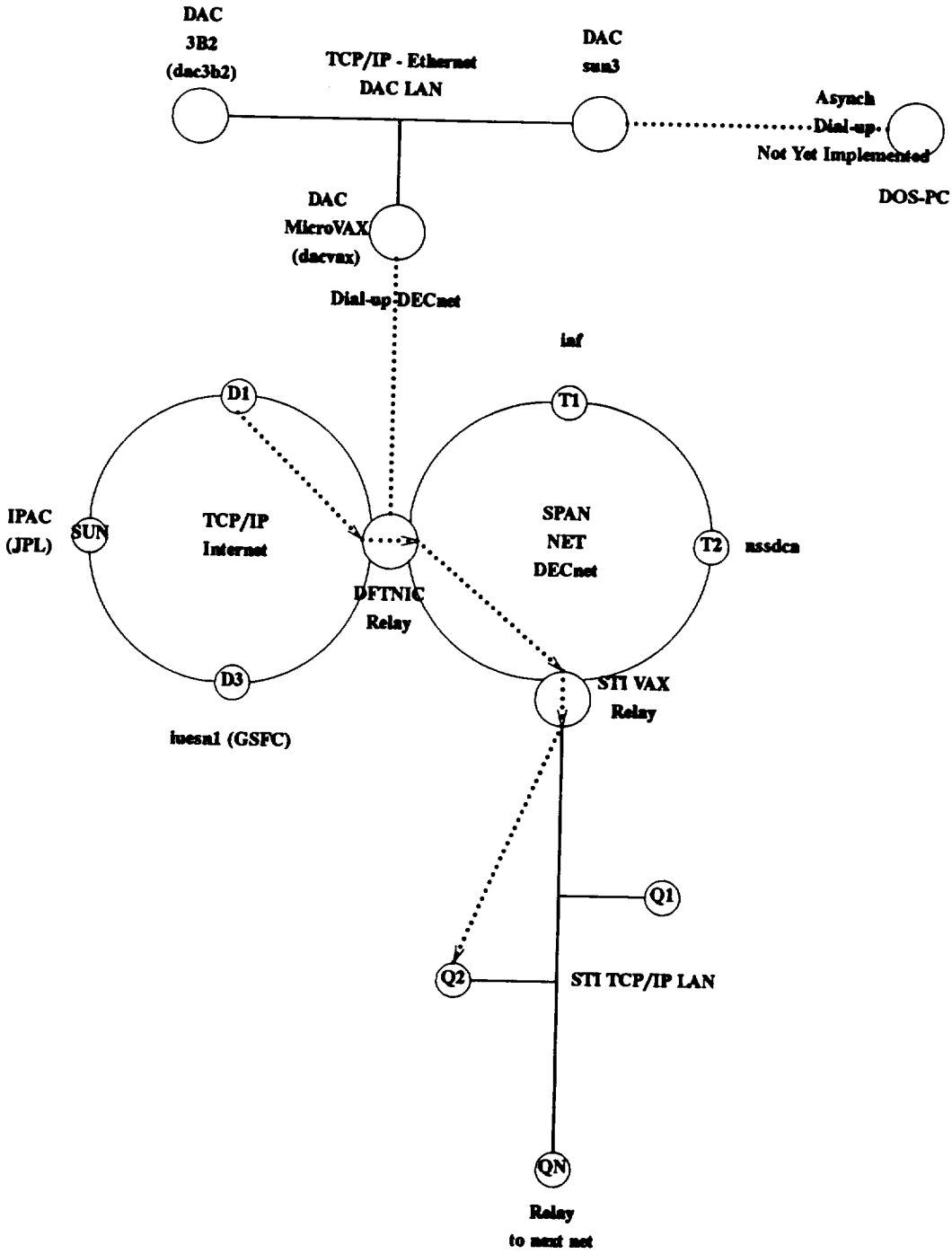
   dnetstat network host -p

   If this operation is successful, the DNET connectionless service is also probably functioning properly.

10. DNET Initial Demonstration Network

10.1 Network Topology

The logical arrangement of the initial DNET demonstration network is shown in the following diagram:



Pilot Network for DNET

Additional details on these sites is provided in the following table:

Tenative DNET Wide Area Demonstration Sites		
Site	**Network(s)**	**Computers**
DAC	DECnet, TCP/IP both Ethernet	MicroVAX, 3B2, HP, PCs
NSSDC- GSFC	DECnet(SPAN) TCP/IP	VAX 8600, 3B2, Sun 3, Sun 4, PCs
NSSDCA - GSFC	DECnet	VAX 8600
IPAC - JPL	TCP/IP	Sun 3
SAO - Cambridge	DECnet	Sun 3, VAX
STI - Balt.	TCP/IP, DECnet	Sun, VAX
IVE - Colo.	TCP/IP, SPAN	Sun, VAX
IVE - GSFC	TCP/IP, SPAN	Sun, VAX

10.2 Information on DNET nodes

A list of current contacts & other information for each DNET node is listed in the file **dnetinfo** which is located in the **dnet_home** directory and part of the usual DNET distribution.

10.3 Starting up (a subset of) the Demonstration Network

Suggested Demonstration Network Subset

1. DAC - brinc - (sun386i) - DAC TCP/IP LAN
2. DAC - dacvax (microvax II) - DAC TCP/IP LAN & Dial-up SPANET (DECnet) connection
3. GSFC - dftnic (vax) - SPANET (DECnet) and Internet (TCP/IP)
4. GSFC - iuesn1 (sun4) - Internet (TCP/IP)

NOTE: A login session must be maintained with each site for the duration of the demonstration. DNET, as described here, requires each a login at each site in order to remain 'up'.

DNET Account information for these machines is given in the file **dnetinfo**, found in the **dnet_home directory.**

1. login to **brinc**
2. enter 'dnstart'
3. On a separate terminal login to **dacvax**
4. Enter the following:

   cd dnet_home

   @dnstart

5. On a separate terminal, login to dacvax as 'system'. Then follow the instructions in the section below on establishing an asynchronous DECNet connection from **dacvax** to SPANET.

6. once the dacvax SPANET connection has been started perform the following steps to start up DNET on DFTNIC

set host dftnic

login to dftnic

cd dnet_home

@dnstart

7. From a separate terminal, login to **dacvax**, then perform the following steps to connect to **iuesn1**.

   set host dftnic

   login to dftnic

   telnet iuesn1

   login to iuesn1

   when logged in to iuesn1, enter 'dnstart'

To stop DNET, enter 'dnstop' on UNIX systems and @dnstop on VAX systems.

11. Asynchronous DECnet connection from dacvax to SPANET

The connection from DAC to the demonstration network shown in the preceding section is currently accomplished via a low-speed asynchronous DECnet connection which must be manually established. This section describes the procedure for starting/stopping this link.

11.1 Starting the Link

Procedure to establish/drop dial-up DECnet link between DACVAX & 'DFTNIC' at NASA-GSFC.

Assumptions:

- Hayes Modem connected to port on VAX
- Phone line to DAC Switch is connected
- Md. tie line available on the DAC switch

1. Logon to VAX as 'SYSTEM'
2. set host/dte tta1
3. atz[CR]
4. If the response is not OK, try following the steps in the shutdown procedure in the next section; link may be hung from an earlier session.
5. atdt91,2869000[CR]
6. Wait for connection
7. In response to Enter Number:

   type 'lafmpp[CR]'
8. When 'Call Complete' msg appears
9. [CR][CR]
10. In response to 'enter class'

    type 'dftnic' [CR]
11. Type [CR][CR] several times, then enter
12. user: ASYNCH
13. password: enter password here
14. wait for message 'DECnet' control returned
15. test by entering 'set host dftnic'; should respond with a username, password

11.2 Stopping the asynch DECnet link

1. login to the dacvax as SYSTEM
2. Disconnect modem (power switch off, then on); yes, it's not pretty, but don't ask questions!
3. cd sys$system
4. run ncp
5. NCP> set circuit tt-0-1 state off
6. NCP> exit

12. DNET Network Utility Commands

12.1 Examining The Status of DNET

DNET provides a general network utility function **dnetstat** which allows the user to determine a variety of information about local or remote DNET nodes. Information which **dnetstat** can obtain for both local and remote nodes includes:

1. Is DNET 'alive' at the Node?
2. The Number of active and inactive DNET Processes (long and short formats; Streaming and/or Connectionless Options)
3. Statistics of DNET Use at the Node
4. DNET Routing Tables at the Node

The general form of the **dnetstat** command is as follows:

dnetstat [dnet_network] [dnet_host] [options]

If the network and host arguments are both omitted, the local host is assumed by default.

If the status of a host on the local DNET network is required, only the **dnet_host** argument is required (local network is understood).

12.2 Testing if DNET is alive

As an introduction to dnetstat, try using the 'ping' option on your local host. This is done by typing

dnetstat -p

If DNET is 'running' on the local machine, the following message will appear:

DNET is ALIVE at dnet_network dnet_host***

This response indicates that

1. At least one DNET PVC Master Server is running on the local node
2. The DNET Datagram Master Server is running on the local node

If DNET is not running normally on your system, the following message will appear

Timed out waiting for response

Now try using dnetstat to 'ping' another DNET host on the local or a distant DNET network.

If this is successful, you are further assured not only is the DNET software running at that host, but also that the DNET datagram service is operating (at least between your machine and the distant host).

12.3 Obtaining Status of DNET Servers

dnetstat may be used to obtain the status of DNET processes at local and remote DNET nodes.

This information may be obtained in the following formats

1. Connection Oriented Services only
2. Connectionless (Datagram) Services only
3. Both Services
4. Short Display Format - types, number avail, and state of servers
5. Long Format - short format info + (Process IDs) and Start/Idle Times

The short listing of server status is shown below. The command used is:

dnetstat [network] [host]

********* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:**

Srv Type	Image	PS	Av	Max	S#
dmstcp					
dechod	dechod	1	1	1	1
drexecd	drexecd	1	1	1	1
dtftpd	dtftpd	1	1	1	1
dncld	dncld	3	3	3	3
dlogind	dlogind	1	1	1	1

********* DNET CONNECTIONLESS (Datagram) STATUS at: dnett1 sun3:**

ProcName	S	StartTime
dgstcp	1	Aug 1 10:44
	1	Aug 1 10:44
dnstatd	1	Aug 1 10:44
dnetstat	1	Aug 1 10:46

A longer listing of the server status may be obtained using the l (long) and c (connection) options.

dnetstat [network] [host] -lcd

```
******* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:
Srv Type    Image      PS   Av   Max   S#   PID    IU   St Time        Idle Since
-------------------------------------------------------------------------------
dmstcp                                       5489        Aug 1 10:44
dechod      dechod     1    1    1     1    5491   N                   Aug 1 10:44
drexecd     drexecd    1    1    1     1    5492   N                   Aug 1 10:44
dtftpd      dtftpd     1    1    1     1    5493   N                   Aug 1 10:44
dncld       dncld      3    3    3     3    5494   N                   Aug 1 10:44
                                             5497   N                   Aug 1 10:44
                                             5498   N                   Aug 1 10:44
dlogind     dlogind    1    1    1     1    5499   N                   Aug 1 10:44
```

A long listing of the both virtual circuit and datagram server status may be obtained using the l (long), c (connection), and d (datagram) options.

dnetstat [network] [host] -lcd

```
******* DNET VIRTUAL CIRCUIT SERVER STATUS at: dnett1 sun3:
Srv Type    Image      PS   Av   Max   S#   PID    IU   St Time        Idle Since
-------------------------------------------------------------------------------
dmstcp                                       5489        Aug 1 10:44
dechod      dechod     1    1    1     1    5491   N                   Aug 1 10:44
drexecd     drexecd    1    1    1     1    5492   N                   Aug 1 10:44
dtftpd      dtftpd     1    1    1     1    5493   N                   Aug 1 10:44
dncld       dncld      3    3    3     3    5494   N                   Aug 1 10:44
                                             5497   N                   Aug 1 10:44
                                             5498   N                   Aug 1 10:44
dlogind     dlogind    1    1    1     1    5499   N                   Aug 1 10:44

******* DNET CONNECTIONLESS (Datagram) STATUS at: dnett1 sun3:
ProcName    S    PID    IPC-Name    IPCID  SIG  MSzStartTime
-------------------------------------------------------------
dgstcp      1    5482   DN_5482     1      0    0Aug 1 10:44
            1    5481   DN_5481     2      0    0Aug 1 10:44
dnstatd     1    5495   DN_5495     3      0    0Aug 1 10:44
dnetstat    1    5504   DN_5504     4      0    0Aug 1 10:45
```

To obtain the routing table at a particular host, enter the following command:

dnetstat [network] [host] -r

An example of output resulting from this command is:

```
******* DNET ROUTING TABLE at: dnett1 sun3:
DestNet     Nxt Host    Nxt Proc    DG Protocol
-------------------------------------------------
dnett1      NULL        NULL        tcp
spanet      dacvax      drelaytd    tcp
starnet     dacvax      drelaytd    tcp
```

12.4 Underlying Processes for Network Status

The **dnetstat** process is invoked on demand at any DNET host. It provides a set of generalized network utility functions. It interacts with the DNET Status Server **dnstatd** located at either the local or any remote DNET node. Its functions are:

1. Determine status of local/remote DNET processes
2. Determine/report status (UP/DOWN) of remote DNET nodes
3. Determine current load of remote DNET processes
4. Update DNET Routing Tables
5. Update of DNET Well Known Server Table (if required)
6. Maintain other DNET status information for use by local processes

The relationship between the Status Client and a Status Server on one of the DNET hosts is shown in the following diagram:

12.4.1 Update Local Routing Table

The hierarchical routing table at a DNET host may be updated through the following procedure:

1. Poll local DNET Gateway(s) for their routing table
2. Retrieve the Gateway Routing tables
3. In turn, poll the more distant gateways to retrieve their routing tables
4. By deduction, determine local routing table contents

13. DNET Errors

The following errors are defined within DNET.

```
#define     D_NOERR      0      /* No DNET error */
#define     D_SYSERR     1      /* A system error has occurred */
#define     D_BADSTATE   2      /* program in wrong state to issue this dnet call */
#define     D_BADARG     3      /* value of argument was determined to be invalid */
#define     D_OVRFLW     4      /* overflow of i/o buffer */
#define     D_AEXIST     5      /* The specified object already exists */
#define     D_ESRVRSP    6      /* Error return value in DGMS service req response */
#define     D_EPERM      7      /* Permission Denied */
#define     D_NOMSG      8      /* D_NOWAIT flag set and no message waiting to be read */
#define     D_NODGRSC    9      /* No more available DGMS resources */
#define     D_INTERN     10     /* Internal DNET error */
#define     D_BADNM      11     /* Invalid process name was specified */
#define     D_DGTB       12     /* Datagram To Big */
#define     D_MSGTB      13     /* Message To Big */
#define     D_BADHN      14     /* Could not find net/host combination in router tables */
#define     D_ADGENF     15     /* ADGUT Entry Not Found */
#define     D_PN2BIG     16     /* Process name string too big */
#define     D_IPCNM2BIG  17     /* IPC name string too big. DNET code error */
#define     D_NOEXIST    18     /* The specified object does not exist */
#define     D_INTR       19     /* A signal interrupted the library routine */
#define     D_NOSRSC     20     /* Temporarily out of system resources */
#define     D_NODNET     21     /* Missing all or part of dnet provider */
#define     D_WOULDBLOCK 22     /* Operation would block */
#define     D_TIMEOUT    23     /* Timeout or retry count exceeded */
#define     D_QUOTA      24     /* Quota limit exceeded */
#define     D_NOSYSFILE  25     /* DNET system file/table not found */
#define     D_SYNERR     26     /* DNET system file/table syntax error */
#define     D_NOIMAGE    27     /* Image (server) not file not found */
#define     D_HOMELESS   28     /* Env variable 'dnet_home' not defined */
#define     D_SRVNOACK   29     /* No respone from application server */
#define     D_NOHOST     30     /* No such host */
#define     D_NOPATH     31     /* DNET could not find a path for the src/dest pair */
#define     D_SYSLIBERR  32     /* System library function failed */
#define     D_NODNETSRV  33     /* DNET servers dms/dgstcp not defined in 'etc/services' */
#define     D_SHUTDOWN   34     /* Orderly shutdown from master server */
#define     D_MAXERRS    35
```

```
static          char            *dgms_errmsgs[D_MAXERRS] = {
"No DNET error",
"A system error has occurred",
"program in wrong state to issue this dnet call",
"value of argument was determined to be invalid",
"overflow of i/o buffer",
"The specified object already exists",
"Error return value in DGMS service req response",
"Permission Denied",
"D_NOWAIT flag set and no message waiting to be read",
"No more available DGMS resources",
"Internal DNET error",
"Invalid process name was specified",
"Datagram To Big",
"Message To Big",
"Could not find net/host combination in router tables",
"ADGUT Entry Not Found",
"Process name string too big",
"IPC name string too big. Probably DNET internal code error",
"The specified object does not exist",
"A signal interrupted the library routine",
"Temporarily out of system resources",
"Missing all or part of dnet provider",
"Operation would block",
"Timeout or retry count exceeded",
"Quota limit exceeded",
"DNET system file/table not found",
"DNET system file/table syntax error",
"Image (server) file not found",
"Env variable 'dnethome' not defined",
"No respone from application server",
"No such host",
"DNET could not find a path for the src/dest pair",
"System library function failed",
"DNET servers dms/dgstcp not defined in '/etc/services'",
"Orderly Shutdown from master server"
};
```

14. DNET Security

14.1 Execution Security

Access to remote DNET hosts is always via the PVC or Datagram Master Servers. Once connected to the remote host, the DNET client process will be connected to the corresponding server, if one is available. An optional **login** function may be placed in any of the DNET client-server pairs (currently login is required for **dlogin** and **dtftp**). See the next section and the DNET Programmer's GUIDE & REFERENCE MANUAL for further information on the use of **dn_login**.

When DNET provides access to remote execution of processes, the execution privileges for non-DNET processes are the same as for a locally-connected user.

14.2 User Security

The functions **dn_login** & **dn_login_verify** may be used on the client and server DNET applications respectively in order to validate user access to the server machine.

14.2.1 UNIX

The **/etc/passwd** is used by dn_login_verify in the UNIX environment to validate DNET users where an application requires such validation. The user login account names and passwords are maintained in the normal fashion for the local UNIX system.

14.2.2 VMS

No password protection is currently implemented on VMS systems for DNET other than a very weak, 'hardwired' password.

A more general approach would incorporate a routine to access the **uaf.dat** file in order that user passwords could be checked. Attempts to locate and/or write such a routine have been unsucessful to date.

14.3 File Security

Access to files on individual systems is dependent on the local file protection mechanisms.

Once a user has been 'validated' using the **dn_login** he has the same file access privileges as he would have had he 'logged' on to that host via some other procedure.

15. Electronic Mail Administration

The initial version of DNET mail is quite elementary in concept and requires no maintenance.

The mail client process **dmail** interacts with local or remote server processes **dmaild** and places mail in a file with the name of the destination user (account) in the directory **dnet_home/mail**.

DNET mail will operate on the local node provided

1. the **dmaild** server is operational (has been started by the DNET master server).
2. the directory **dnet_home/mail** exists - this is ordinarily created by the DNET **postmove** procedure when DNET is installed on a local machine.

If mail fails to operate, these two conditions should be checked and appropriate action taken as needed.

16. Library and Program Pool Administration

Master source code for DNET is maintained under sccs on an AT&T 3B2-600 at DAC in the directory **/usr/nasa/dnet/sccs_src**. The 'master' copies all DNET code are maintained in under this top directory with the following organization. The 'standard' DNET directory structure is shown in the figure below:

usr/nasa/dnet/sccs_src/ ..

/common /pvcdir /dgdir /appdir

NOTE: Administration and maintenance of the files in this directory tree is essential for the integrity of the DNET code.

Most common modifications to DNET will likely occur in files in ../sccs_src/ and st Changes to the subdirectories ../common, ../dnet/pvcdir, ../dnet/dgdir should only be undertaken with a view toward global changes in mind. The administrator of these files should make every effort to ascertain that the DNET code at all sites 'in the field' is consistent with the latest copy maintained under sccs and vice versa.

17. DNET Performance Monitoring

17.1 General

There was no performance specification for throughput or system loading for this initial version of DNET. Nevertheless, some care was exercised in both the design and implementation of the software in an attempt to make the overhead of DNET as low as possible. These efforts were qualitative in nature, empirical attempts to make DNET have as 'light' an effect as possible on the systems and networks involved.

Despite the absence of a formal DNET performance specification, it seemed useful to provide at least some rudimentary quantititative performance measurements for the system. This section describes the measurements made on DNET responsiveness.

In a distributed, heterogeneous environment, it can be exceedingly difficult to define a 'stable' test environment for conducting timing tests of any kind. The number of users, number and types of processes, and the network communication traffic in the test environment may all be beyond the control of the person performing the test.

At this stage of development, it seems most important to comment on how DNET compares with a comparable homogeneous network environment under similar conditions. This can be done in the DAC portion of the DNET testbed environment.

17.2 DNET Performance Test Application - dptc

17.3 VMS Host vs UNIX Host

17.4 DECnet vs TCP/IP

18. Glossary

The following terms are used in the description of DNET:

Applications Servers-

Servers such as File Transfer, Remote Login, Remote Execution, etc. that perform services for clients. Applications Servers are invoked on demand by clients after using the Service Assignment to obtain the name of an available server.

Connection Lock Table-

List of open connections held by process for use by its Basic Datagram I/O package. Locked connections result from user requests for Permanent Virtual Circuits.

Datagram Master Server (DGMS)-

A server process, located at each DNET host and gateway, which provides an interface to DNET clients and servers and the DNET Connectionless Datagram and Signalling Service

Datagram Protocol Servers (DPS)-

Protocol specific servers located at each DNET host and gateway, which provides an DNET Connectionless an interface to the underlying network Datagram service.

Master Server Init Table-

These tables, **tbls.msinittcp** and **tbls.msinitdec** contain initialization information for the DNET Master Servers including the type of server to be activated, the maximum # allowed at this host, and the number to make available initially, and an indication of whether the server must be prespawned. The tables are updated by the local System Administrator at the specific DNET host.

Master Server Table-

One for each DNET host, it contains information on the types and numbers of each class of DNET server actively supported on this node at any instant. Each generic server entry points to a **Server Instance Table** which lists the current specific instances of a particular class of server. It is updated by the Master Server and by specific DNET application servers.

Master Server Process (DMS)-

Processes, one per Network, managing the Master Server Table, handling server registration, server assignment, and server control. They are spawned by network start-up command files.

DNET Basic I/O package-

Included as library within an application program, it provides network i/o interface including datagram formatting.

Gateway-

A DNET node at which communicaton protocol boundary is passed. DNET relay servers move data from one network to another performing an effective protocol conversion for streaming services. These servers are created, allocated, and used like any other DNET streaming applications servers. The Datagram Master Server, in conjunction with protocol specific datagram servers performs a similar function for DNET datagrams.

Network Command Line Interpreter-

DNET Client process that directs the execution of network commands using datagrams sent to various hosts and several servers.

myname - hostname table-

A table, **tbls.myname**, maintained in the **dnet_home** directory on each DNET node lists the DNET networks to which that host is connected and the name(s) by which the local host is known on those networks.

Network Command Language Processor-

Server that directs the execution of network commands using datagrams sent to various hosts and several servers. It is an application server, copies can be pre-spawned or spawned on demand.

Network Command Server-

Spawned by request from Command Language Processor, this Server is directed by Command Language Processor. It spawns processes and directs i/o to execute network commands.

Network Status Server-

Resides in each network host. Receives Host Status Tables, Host Alias Table, Well Known Server Tables, Connectivity Tables, and periodically sends "I am alive" messages to the Administrative host. To ensure these periodic messages are sent the Basic datagram I/O package uses a timer/wake-up signal to initiate the transmission of the message to the Network Status Client. Because this is done by the I/O package and there is a copy of this package in every process that uses network I/O the network status data is collected on a per process not per host basis.

PVC Relay

A DNET relay used in the completion of DNET Permanent Virtual Circuits (PVCs).

Relay

Special DNET application processes located in a DNET gateway which perform protocol conversion for DNET streaming service between dissimilar networks. The appropriate Master Server process 'listens' on a particular protocol boundary when

idle and assigns a relay when a request for a protocol h'hop' is received from DNET.. The relays are named according to the protocol boundary which they are intended to bridge. Thus a T-D relay services requests which arrive on a TCP/IP network, relaying data to a DECnet net. Relays operate in a full duplex mode while actually in use.

Router

DNET employs a hierarchical routing strategy. Each DNET node has, for every (DNET) network known to it, information on the next DNET host to contact in order to move data toward the destination. The DNET router function uses the information in the routing table as follows: Given a destination network, host, and process, returns the next 'best' hop (network, host, process) to 'move' toward the destination.

Routing Table-

A hierarchical routing table that contains the next 'hop' from the local DNET host/network in the direction of all other DNET networks. A minimal version of this table is provided with the distribution copy of DNET. The table is currently maintained manually by the local system administrator. In the future, this table will be dynamically configured and maintained by the local DNET Network Status Server after intial startup has taken place. The routing table is named **tbls.net** and is located in the **dnet_home** directory.

Server Assignment Function-

Returns the name of an available server to a requesting Router, and updates the Domain Server Table.

Server Instance Table(s-

Lists the current specific instances of a particular class of DNET Application Server. Entries are made by the Master Server and cleared via dn_done() calls from the servers as they complete their tasks.

Server Registration Function-

This function is part of the Domain Server Process. It updates the Domain Server table with information from Servers (e.g."now in use").

DNET

ADMINISTRATOR' S REFERENCE

Version: 1.14
Print Date: 08/31/89 12:59:53
Module Name: admin.ref

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

NAME

checkdmail - dnet 'mail' auto-user notification process

SYNOPSIS

checkdmail

DESCRIPTION

The checkdmail process is invoked by the DNET login script when a user logs in to a DNET host. It checks if a non-null mail file is present in the directory **dnet_home/mail** (dnet_home:mail for VMS), and if so notifies the user with the message:

You have DNET mail .....

The user may then invoke the **dmail** client to read this mail file.

The ability to run checkdmail depends on its presence in the Master Server Init Table for the destination host.

SEE ALSO

dms, dmail(1D), dmaild(1D)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

dechod - dnet 'echo' server

SYNOPSIS

Must be entered into the dms init table.

DESCRIPTION

The dechod 'echoes' command lines sent to it from a distant DNET decho client process back to that client.

dechod is started by the local DNET Master Server according to information in the Master Server Init Table. A DNET permanent virtual (streaming) connection is opened to the destination network:host. Command line input at the local host is then echoed back from the destination after each carriage return.

In conjuction with **decho** dechod provides a convenient means of demonstrating the setup time and end-to-end performance of the DNET streaming service.

SEE ALSO

decho(1), tbls.msinitxxx(4)

DIAGNOSTICS

If the dnet_debug environmental variable (logical name in VMS) is defined with a non-zero value, then a log file will be generated where error output may be viewed. If the dnet_debug value is zero, then all error output will be discarded.

NAME

dlogind - dnet 'remote execution' server

SYNOPSIS

dlogind

DESCRIPTION

dlogind is the DNET server used to provide remote login function over the DNET network.

The ability to run dlogind depends on its presence in the Master Server Init Table for the destination host.

SEE ALSO

dms, dlogin(1), drexec(1)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

dmaild - dnet 'mail' server

SYNOPSIS

dmaild

DESCRIPTION

dmaild acts a simple mail transfer server from a remote cmail client.

The ability to run dmaild depends on its presence in the Master Server Init Table for the destination host.

SEE ALSO

dms, dmail(1D), checkdmail(1)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

dncl - dnet 'network command language' server

SYNOPSIS

dncl

DESCRIPTION

The **dncl** command invokes the interactive dnet network command language program. This program allows for processing of a single data stream in a distributed environment. To do this, the processing of the data stream is broken into sub command lines SCL (which together make up the dncl command line CL). The dncl CL may be entered after the dncl prompt:

dncl >

The following is a synopsis of the dncl command line:

SCL > SCL [> SCL] ...

You will note that a minimum of two SCL components are required in a CL. The reason for this will be explained when we look at the three categories of SCL components. Also note that the > symbol is used to delimit the SCL components.

The following is a synopsis of the SCL component:

[[netname::]hostname:][*]command/file

Notice that **netname** and **hostname** are optional, although if a network name is supplied, then a host name must also be supplied. In the case where both **netname** and **hostname** are specified, a double colon must delimit the **netname** and the **hostname**, and a single colon must delimit the **hostname** and the **command/file**. Further, if the **command/file** value contains a colon, then the **hostname** must be supplied at a minimum so that the colon within the **command/file** will be ignored by dncl.

If the requested node is the current machine (the **netname** and **hostname** combination represent the current machine), and no colons appear within the **command/file** value, then **netname** and **hostname** may be omitted. Similarly, if the **hostname** machine is on the current network, then **netname** may be omitted. On dnet gateway machines remember that only one network is considered to be current. This means that although the network may be directly connected to the current machine, it can not be considered a current network.

The **command/file** portion of the SCL represents either a file or a command to be accessed on the given machine and falls into one of three categories:

- First SCL component -- must be a file
- Middle SCL component -- must be a command (precede with *)
- Last SCL component -- must be a file

As you will remember from the CL synopsis above, and minimum of two SCL components must be specified (a First SCL component and a Last SCL component). This represents the simplest form of a dncl CL and results in a file transfer without filtering. The dncl CLs of greater complexity merely represent a higher degree of filtration between the first and last SCL components. The filtration described here is provided by the middle SCL component category (a command). This command is assumed to read input from a standard location, process the input received and generate output to a standard location. Many commands can be described in this fashion (input/processing/output), but not all work with standard locations for input and output. Commands that do use standard locations and work in the input/processing/output fashion are described as being filters. To work properly as a middle SCL category SCL component, the command must also be a filter, as unpredictable results will otherwise occurr.

All middle SCL category SCL components must be preceded with an asterix (*) as shown in the SCL synopsis above.

The UNIX operating system is rich with existing filters to perform a variety of tasks. These filters are comparatively rare in the VMS operating system. Despite this, filters may be created for VMS with C language programs by using the predefined **stdin** and **stdout** streams with the standard I/O package.

SEE ALSO

dtftp(1), dsh(1)

RETURN VALUE

After successful completion of a dncl CL, the following message will be displayed:

ACKCOMP received.

This means that the ACKCOMP (ACKnowledge COMPletion) packet has been initiated by the last SCL category driver, and has been successfully passed back through all intermediate SCL components to be successfully received by the dncl command invoked by the user.

If the ACKCOMP received message is not displayed, then a cryptic error message will be displayed describing the reason for failure. If the error message is preceded by **dncld:**, then this means that the error occurred at a possibly remote node, and this message was propagated back to be viewed by the user.

A common form of error message is:

No route to netname::hostname:dncld

This indicates that the node specified could not be found from the current location. Two things should be remembered to help to solve this problem:

1. You may not have specifed the node name portion of the stated SCL, and the default may have been used.
2. The node is always relative to the node on the previous SCL component. The first SCL is always relative to your current node. As an example, if the first SCL was specifed as: **spanet::iaf:sys$login:myfile** , and the second SCL was: ***sort**, then it would try to spawn the sort filter on the spanet::iaf node.

CAVEATS

Never make assumptions about current location within a file system on any node when creating SCL components. Absolute pathnames or logical names must be used for files. For commands, absolute pathnames or logical names must also be used, but on UNIX operating systems, the PATH environmental variable may be set by the dnet administrator before the dncl drivers are initiated so that they can be forced to look in non-normal locations for UNIX filters.

NAME

dnetstat - obtain dnet network status

SYNOPSIS

dnetstat [*dnet_network*] [*dnet_host*] **[-acdfhlnprs]**

DESCRIPTION

The **dnetstat** command allows the display of various DNET-related data structures. Information may be displayed in various forms, depending on the option which is specified. **dnetstat** can be used to determine the status of all DNET servers, routing tables, and server usage statistics.

Information may be displayed for the local DNET node or may be retrieved and displayed for other DNET nodes.

Options:

dnet_network - the DNET network of the DNET host from which information is desired; if omitted, local network is assumed

dnet_host - the DNET network of the DNET host from which information is desired; if both network and host omitted, local host is assumed

If none of the below options is specified, the defaults **local_host** & **[-cd]** are assumed

-a Display all available information (in long format)

-c Display Status of Connection (Streaming) Servers

-d Display Status of Datagram (Connectionless) Servers

-f Display PIDs, etc. in alternate (Decmimal/Hexidecimal) format; allows optional conversion between machines with different display formats

-h Display help on options for **dnetstat**

-l Display other specified options in **long** or extended format

-n show DNET map (network, host)

-p ping the specifed host - i.e. test if DNET is alive on the specified host **p** overrides all other options. If successful, the message:

DNET is Alive at dnet_network dnet_host

is printed on the terminal If the 'ping' operation is unsuccessful, **dnetstat** will usually timeout waiting for the response from **dnstatd.**

Timed out waiting for response

-r show DNET routing tables for the specified node

-s show per-DNET server statistics (dtftp, drexec, dmail, dncl)

SEE ALSO

dnstatd, tbls.msinitdec, tbls.msinitdec, tbls.net

DIAGNOSTICS

The call fails if:

Specified host is not up

DNET is not operating on the specified host

dnstatd is not operating on the specified host

In each of the above instances, **dnetstat**, will report:

Timed out waiting for response

NAME

dnlogin.csh - DNET logir script for UNIX systems with 'C' Shell

SYNOPSIS

@dnlogin.csh

DESCRIPTION

The **dnlogin.csh** script sets up the operating environment for DNET when a user logs in to a UNIX system running the 'C' Shell. dnlogin.csh is ordinarily invoked from the file **.prolile** in the user's login directory. A listing of the script follows:

```
# dnlogin.csh
# login script for C Shell under UNIX for DNET
# First set dnet_home in your .login.
# Then source this file.
setenv PATH "$PATH":$dnet_home/bin
checkdmail
```

NAME

dnlogin.dft - DNET login script for NASA-GSFC DFTNIC VAX

SYNOPSIS

@dnlogin.dft

DESCRIPTION

The **dnlogin.dft** script sets up the operating environment for DNET when a user logs in to the DFTNIC VAX running VMS. dnlogin.dft is ordinarily invoked from the file **login.com** in the user's login directory. A listing of the script follows:

```
$! dnlogin.dft
$! login script for DNET for dftnic
$
$! logical names
$
$ DNET_DEBUG = = "0"
$
$ define/job dnet_pvcdir $cldata:[dnet.dnet.pvcdir]
$ define/job dnet_dgdir $cldata:[dnet.dnet.dgdir]
$ define/job dnet_common $cldata:[dnet.dnet.common]
$ define/job dnet_appdir $cldata:[dnet.dnet.appdir]
$! define/job dnet_debug 1
$ set proc/priv=grpnam
$ define/group dnet_home $cldata:[dnet.dnet]
$ define/group dnet_bin $cldata:[dnet.dnet.bin]
$ define/group dnet_gateway 1
$
$ ptar = = "$ cldata:[dnet.bin]ptar.exe"
$
$ define c$include dnet_common, dnet_pvcdir, dnet_dgdir, exos_etc
$ define vaxc$include c$include, sys$library
$
$! Clients
$
$ decho  = = "$ dnet_bin:decho.exe"
$ ddechoc  = = "$ dnet_bin:ddechoc.exe"
$ dechon  = = "$ dnet_bin:dechon.exe"
$ drexec  = = "$ dnet_bin:drexec.exe"
$ dtftp  = = "$ dnet_bin:dtftp.exe"
$ dlogin = = "$ dnet_bin:dlogin.exe"
$ dmskill = = "$ dnet_bin:dmskill.exe"
$ dnetstat = = "$ dnet_bin:dnetstat.exe"
$ dncl = = "$ dnet_bin:dncl.exe"
$ dmail = = "$ dnet_bin:dmail.exe"
```

```
$
$! development only
$
$! delmbx   = = "$ $disk1:[odnet]delmbx.exe"
$ shack = = "$ dnet_bin:shack.exe"
$ i_to_o = = "$ dnet_bin:i_to_o.exe"
$ bcd = = "$ dnet_bin:bcd.exe"
$ ddechoc = = "$ dnet_bin:ddechoc.exe"
$
$! aliases
$
$ sl = = "show logical"
$ ss = = "show symbol"
$ ls = = "dir"
$ l = = "dir"
$ cd = = "set def"
$ pwd = = "show def"
$ vi = = "ed"
$ view = = "ed/read_only"
$ ps = = "show system"
$ ns = = "netstat -a"
$ more = = "type/page"
$ clear = = "@clear"
$
$ set proc/priv=sysnam
$
$ cd dnet_home
$ checkdmail
$
```

NAME

dnlogin.dv - DNET login script for DAC Microvax II

SYNOPSIS

@dnlogin.dv

DESCRIPTION

The **dnlogin.dv** script sets up the operating environment for DNET when a user logs in to a VAX running VMS. dnlogin.dv is ordinarily invoked from the file **login.com** in the user's login directory. A listing of the script follows:

```
$! dnlogin.com
$! login script for DNET for dftnic
$
$! logical names
$
$ DNET_DEBUG = = "0"
$
$ define/job dnet_pvcdir $cldata:[dnet.dnet.pvcdir]
$ define/job dnet_dgdir $cldata:[dnet.dnet.dgdir]
$ define/job dnet_common $cldata:[dnet.dnet.common]
$ define/job dnet_appdir $cldata:[dnet.dnet.appdir]
$! define/job dnet_debug 1
$ set proc/priv=grpnam
$ define/group dnet_home $cldata:[dnet.dnet]
$ define/group dnet_bin $cldata:[dnet.dnet.bin]
$ define/group dnet_gateway 1
$
$ ptar = = "$ cldata:[dnet.bin]ptar.exe"
$
$ define c$include dnet_common, dnet_pvcdir, dnet_dgdir, exos_etc
$ define vaxc$include c$include, sys$library
$
$! Clients
$
$ decho  = = "$ dnet_bin:decho.exe"
$ ddechoc  = = "$ dnet_bin:ddechoc.exe"
$ dechon  = = "$ dnet_bin:dechon.exe"
$ drexec  = = "$ dnet_bin:drexec.exe"
$ dtftp  = = "$ dnet_bin:dtftp.exe"
$ dlogin = = "$ dnet_bin:dlogin.exe"
$ dmskill = = "$ dnet_bin:dmskill.exe"
$ dnetstat = = "$ dnet_bin:dnetstat.exe"
$ dncl = = "$ dnet_bin:dncl.exe"
$ dmail = = "$ dnet_bin:dmail.exe"
$
```

```
$! development only
$
$! delmbx   = = "$ $d:sk1:[odnet]delmbx.exe"
$ shack = = "$ dnet_bin:shack.exe"
$ i_to_o = = "$ dnet_bin:i_to_o.exe"
$ bcd = = "$ dnet_bin:bcd.exe"
$ ddechoc = = "$ dnet_bin:ddechoc.exe"
$
$! aliases
$
$ sl = = "show logical"
$ ss = = "show symbol"
$ ls = = "dir"
$ l = = "dir"
$ cd = = "set def"
$ pwd = = "show def"
$ vi = = "ed"
$ view = = "ed/read_only"
$ ps = = "show system"
$ ns = = "netstat -a"
$ more = = "type/page"
$ clear = = "@clear"
$
$ set proc/priv=sysnam
$
$ cd dnet_home
$ checkdmail
$
```

NAME

dnlogin.sh - DNET login script for UNIX systems with Bourne Shell

SYNOPSIS

@dnlogin.sh

DESCRIPTION

The **dnlogin.sh** script sets up the operating environment for DNET when a user logs in to a UNIX system running the Bourne Shell. dnlogin.sh is ordinarily invoked from the file **.prolile** in the user's login directory. A listing of the script follows:

```
# dnlogin.sh
# login script for Bourne shell under UNIX for DNET
# First set dnet_home in your .profile.
# Then . this file.
PATH=$PATH:$dnet_home/bin
export PATH
checkdmail
```

NAME

dnstart - start local DNET node

SYNOPSIS

dnstart

DESCRIPTION

The **dnstart** command allows the system administrator on a UNIX host to start-up the local DNET node.

SEE ALSO

dnstart.com(8), dnstop(8), dnstop.com(8), startdgms

DIAGNOSTICS

The dnstart module is a UNIX shell script. The output directly from the script merely informs the user of the modules being started up. Output from those modules may also appear on the screen. The script in turn calls the **startdgms** to start the DNET datagram service.

The dnstart script is listed below:

```
dnstop
dnet_home=`echo "${dnet_home}/" | tr -s '/' '/'`
dnet_log = "/tmp/dnet/";export dnet_log
rm -rf /tmp/dnet
mkdir /tmp/dnet
ipcrm -Q 100
cd $dnet_home/bin
startdgms
sleep 2
echo '**********Starting dmstcp**********'
dmstcp &
sleep 1
echo '**********Starting dnstatd**********'
dnstatd &
pidlist=$pidlist" "$!
echo $pidlist > > "${dnet_home}pidlist"
sleep 5
echo '**********DNET now running!**********'
```

NAME

dnstart.com - start local DNET node (on VAX systems)

SYNOPSIS

@dnstart

DESCRIPTION

The **dnstart.com** command allows the system administrator on a UNIX host to start-up the local DNET node.

SEE ALSO

dnstart(8), dnstop(8), dnstop.com(8), strdgms.com, strstat.com

DIAGNOSTICS

The dnstart.com file is a VMS DCL shell script. The output directly from the script merely informs the user of the modules being started up. Output from those modules may also appear on the screen. The script in turn calls the **strdgms.com and strstat.com** to start the DNET datagram service and the network status server respectively.

The dnstart.com script is listed below:

```
$
$ cd dnet_home
$
$ @dnstop
$
$ write sys$output "deleting old log files"
$ del *.log;*
$ write sys$output "purging dnet_home"
$ purge
$
$ cd [.bin]
$
$ write sys$output "purging dnet_home"
$ purge
$ cd dnet_home
$
$ write sys$output "starting dnet DGMS ..."
$ @strdgms
$
$ wait 00:00:05.00
$ @strpvc
$
$ wait 00:00:05.00
$ @strstat
$
$ cd dnet_home
$
$ wait 00:00:05.00
$ write sys$output "DNET Successfully STARTED ********"
$
```

NAME

dnstart.det - start local DNET node (on VAX systems) in a detached mode

SYNOPSIS

@dnstart.det

DESCRIPTION

The **dnstart.det** command allows the system administrator on a UNIX host to start-up the local DNET node in a detached mode. The effect of running 'detached' is to allow DNET to remain operational without DNET being 'logged in'.

SEE ALSO

dnstart(8), dnstop(8), dnstop.com(8), strdgms.com, strstat.com

DIAGNOSTICS

The dnstart.det file is a VMS DCL shell script. The output directly from the script merely informs the user of the modules being started up. Output from those modules may also appear on the screen. The script in turn calls the **strdgms.com and strstat.com** to start the DNET datagram service and the network status server respectively.

The dnstart.det script is listed below:

```
$ ! dnstart.det - start DNET in detached mode
$
$ cd dnet_home
$
$ ! stop any existing DNET processes
$ @dnstop
$
$ cd [.bin]
$
$ set proc/priv=grpnam
$ define/group dnet_home "$disk1:[sys0.dnet.dnet]"
$ define/group dnet_bin "$disk1:[sys0.dnet.dnet.bin]"
$ define/group dnet_gateway 1
$
$ write sys$output "starting dgms ...DETACHED"
$ spawn/in=nl:/out=dnet_home:dgms.log/process=dgms/nowait -
  run/detached/nodebug/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=1200/page_file=30000 -
  /privileges=(grpnam,sysnam,netmbx,tmpmbx)/proc=dgms dgms
$
$ write sys$output "starting dgsudp_in ...DETACHED"
$ run/detached/nodebug/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=1200/page_file=30000 -
  /privileges=(grpnam,netmbx,tmpmbx)/proc=dgsudp_in dgsudp_in
$
$ write sys$output "starting dgsudp_out ...DETACHED"
$ run/detached/nodebug/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
```

```
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=1200/page_file=30000 -
  /privileges=(grpnam,netmbx,tmpmbx)/proc=dgsudp_out dgsudp_out
$
$ write sys$output "starting dgsdec ...DETACHED"
$ run/detached/nodebug/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=1200/page_file=30000 -
  /privileges=(grpnam,sysnam,netmbx,tmpmbx)/proc=dgsdec dgsdec
$
$ wait 00:00:05.00
$
$ write sys$output "starting dmsdec ...DETACHED"
$ run/detached/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=1200/page_file=30000 -
  /privileges=(sysnam,netmbx,tmpmbx)/proc=dmsdec dmsdec
$
$ write sys$output "starting dmstcp ...DETACHED"
$ run/detached/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=12000/page_file=30000 -
  /privileges=(sysnam,netmbx,tmpmbx)/proc=dmstcp dmstcp
$
$ wait 00:00:05.00
$
$ write sys$output "starting dnstatd ...DETACHED"
$ run/detached/buffer_limit=60000/subprocess_limit=30/io_buffered=20 -
  /maximum_working_set=1024/extent=1024/working_set=512/queue_limit=30 -
  /ast_limit=30/file_limit=200/job_table_quota=12000/page_file=30000 -
  /privileges=(sysnam,netmbx,tmpmbx)/proc=dnstatd dnstatd
$ cd dnet_home
$
```

NAME

dnstatd - DNET network status server

SYNOPSIS

dnstatd

DESCRIPTION

dnstatd is a general network utility which allows the display of various DNET-related data structures. Information may be displayed in various forms, depending on the option which is specified. **dnstatd** can be used to determine the status of all DNET servers, routing tables, and server usage statistics.

Information may be displayed for the local DNET node or may be retrieved and displayed for other DNET nodes.

Options:

dnet_network - the DNET network of the DNET host from which information is desired; if omitted, local network is assumed

dnet_host - the DNET network of the DNET host from which information is desired; if both network and host omitted, local host is assumed

If none of the below options is specified, the defaults **local_host** & **[-cd]** are assumed

-a Display all available information (in long format)

-c Display Status of Connection (Streaming) Servers

-d Display Status of Datagram (Connectionless) Servers

-f Display PIDs, etc. in alternate (Decmimal/Hexidecimal) format; allows optional conversion between machines with different display formats

-h Display help on options for **dnetstat**

-l Display other specified options in **long** or extended format

-n show DNET map (network, host)

-p ping the specifed host - i.e. test if DNET is alive on the specified host **p** overrides all other options. If successful, the message:

DNET is Alive at dnet_network dnet_host

is printed on the terminal If the 'ping' operation is unsuccessful, **dnetstat** will usually timeout waiting for the response from **dnstatd**.

Timed out waiting for response

-r show DNET routing tables for the specified node

-s show per-DNET server statistics (dtftp, drexec, dmail, dncl)

SEE ALSO

dnstatd, tbls.msinitdec, tbls.msinitdec, tbls.net

RETURN VALUE

ERRORS

The call fails if:

Specified host is not up

DNET is not operating on the specified host

dnstatd is not operating on the specified host

In each of the above instances, **dnetstat**, will report:

Timed out waiting for response

NAME

dnstop - stop the local DNET services

SYNOPSIS

dnstop

DESCRIPTION

The **dnstop** command allows the system administrator to stop the local DNET services.

SEE ALSO

dnstart(8)

DIAGNOSTICS

The contents of dnstop are listed below:

```
echo '**********Stopping TCP Master Server **********'
dmskill tcp
stopdgms
```

NAME

dtftpd - dnet trivial file transfer server

SYNOPSIS

dtftpd

DESCRIPTION

The dtftpd is the DNET file transfer server. It provides for the transfer of files to and from remote DNET machines.

SEE ALSO

dms, dtftp(1)

RETURN VALUE

ERRORS

The call fails if:

[D_DGTB]

NAME

makemove - generate a generic image of the DNET source files for transport to a remote machine

SYNOPSIS

makemove

DESCRIPTION

The **makemove** command allows the system administrator at the Master DNET node to generate 'ptar' files of the DNET source code for transport to remote locations.

SEE ALSO

ptar(1)

DIAGNOSTICS

NAME

postmove - generate DNET on a target machine

SYNOPSIS

postmove [-m] [-s] [-sc] [-h]

DESCRIPTION

The **postmove** command generates the DNET source and/or executables on a target DNET host.

-m	make DNET after source code has been unpacked
-sc	suppress cleanup - do not remove local copies of 'ptar' files after executables have been made
-s	create shell environment
-h	help

SEE ALSO

makemove(8), sdenv(1), ptar(1)

DIAGNOSTICS

The postmove utility is a shell program (or DCL script). Short messages will be generated during normal operation informing the user what portion of the postmove is being performed currently. Most error conditions result in an abort with the message indicating that an abort is taking place. A final message will be issued when the postmove has run successfully.

NAME

ptar - pack/unpack file(s) in a generic tar format

SYNOPSIS

ptae [-xct] *filename*

DESCRIPTION

The **ptar** command packs or unpacks a file or files into a generic 'tar' format for shipment to remove DNET target machines.

-x	extract files
-c	Create a ptar file
-t	Table cf contents on existing ptar file

SEE ALSO

makemove(8), postmove(8)

DIAGNOSTICS

The ptar displays a list of file names as they are being extracted or archived.

NAME

startdgms - start local DNET Datagram Service

SYNOPSIS

startdgms

DESCRIPTION

The **startdgms** command allows the system administrator on a UNIX host to start-up the local DNET Datagram service. **startdgms** is ordinarily invoked automatically by the **dnstart** script.

SEE ALSO

dnstart(8), startdgms

DIAGNOSTICS

The startdgms script is listed below:

```
if [ "$dnet_home" = "" ]
then
        echo '    'dnet_home not set...Aborting
        exit 1
fi
dnet_home=`echo "${dnet_home}/" | tr -s '/' '/'`
dnet_log = "/tmp/dnet/";export dnet_log
nohup dgms > "${dnet_log}dgms.log" 2>&1 &
pidlist=$!
odgms="${dnet_log}dgms.log"; export odgms
sleep 1
nohup dgsudp_in > "${dnet_log}dgsudp_in.log" 2>&1 &
pidlist=$pidlist" "$!
odgsin="${dnet_log}dgsudp_in.log";export odgsin
nohup dgsudp_out > "${dnet_log}dgsudp_out.log" 2>&1 &
pidlist=$pidlist" "$!
odgsout="${dnet_log}dgsudp_out.log";export odgsout
sleep 1
echo '**********dgms.log**********'
cat $odgms
echo '**********dgstcp_in.log**********'
cat $odgsin
echo '**********dgstcp_out.log**********'
cat $odgsout
# nohup pingerd > "${dnet_log}pingerd.log" 2>&1 &
# pidlist=$pidlist" "$!
# opingerd="${dnet_log}pingerd.log";export opingerd
# nohup abc > "${dnet_log}abc.log" 2>&1 &
# pidlist=$pidlist" "$!
# oabc="${dnet_log}abc.log";export oabc
# sleep 1
# echo '**********PINGERD.log**********'
# cat $opingerd
# echo '**********ABC.log**********'
# cat $oabc
echo $pidlist > "${dnet_home}pidlist"
```

NAME

strdgms.com - start local DNET node (on VAX systems)

SYNOPSIS

@strdgms

DESCRIPTION

The **strdgms.com** command allows the system administrator on a UNIX host to start-up the DNET datagram service. Ordinarily, this script is executed automatically by the **dnstart.com** script.

SEE ALSO

dnstart(8), dnstop(8), dnstop.com(8), strdgms.com, strsta..com

DIAGNOSTICS

The strdgms.com script is listed below:

```
$ ! strdgms - start DNET Datagram Service
$
$ cd dnet_home
$ cd [.bin]
$
$ set proc/priv=sysnam
$ spawn/in=nl:/out=dnet_home:dgms.log/process=dgms/nowait run/nodebug dgms
$ set proc/priv=nosysnam
$ spawn/in=nl:/out=dnet_home:dgsudp_in.log/process=dgsudp_in/nowait run/nodebug dgsudp_in
$ spawn/in=nl:/out=dnet_home:dgsudp_out.log/process=dgsudp_out/nowait run/nodebug dgsudp_out
$ set proc/priv=sysnam
$ spawn/in=nl:/out=dnet_home:dgsdec.log/process=dgsdec/nowait run/nodebug dgsdec
$! set proc/priv=nosysnam
$! spawn/in=nl:/out=dnet_home:pingerd.log/process=pingerd/nowait run/nodebug pingerd
$! spawn/in=nl:/out=dnet_home:abc.log/process=abc/nowait run/nodebug abc
$
$ cd dnet_home
$
$ type dgms.log
$ type dgsudp_in.log
$ type dgsudp_out.log
$! type pingerd.log
$! type abc.log
```

NAME

dnstop.com - stop the local DNET services on VAX systems

SYNOPSIS

@dnstop

DESCRIPTION

The **dnstop** command allows the system administrator to stop the local DNET services.

SEE ALSO

dnstart.com(8)

DIAGNOSTICS

The contents of dnstop.com are listed below:

```
$
$ cd dnet_home
$
$ write sys$output "stopping dgms....."
$ @stpdgms
$ write sys$output "stopping dmsdec ....."
$ stop dmsdec
$ write sys$output "stopping dmstcp ....."
$ stop dmstcp
$ write sys$output "stopping dnstatd ....."
$ stop dnstatd
$
$ cd dnet_home
$
$ wait 00:00:05.00
$
$ write sys$output "DNET is halted ....."
$
```

NAME

stopdmgs - stop the local DNET datagram service on a UNIX system

SYNOPSIS

stopdgms

DESCRIPTION

The **stopdgms** command allows the system administrator to stop the local DNET services.

SEE ALSO

dnstart(8), dnstop(8), startdgms(8)

DIAGNOSTICS

The contents of stopdgms are listed below:

```
if [ "$dnet_home" = "" ]
then
        echo '    'dnet_home not set...Aborting
        exit 1
fi
dnet_home=`echo "${dnet_home}/" | tr -s '/' '/'`
if [ -f "${dnet_home}pidlist" ]
then
        echo Cleaning up dgms
else
        echo No dgms running
        exit 0
fi
pidlist=`cat "${dnet_home}pidlist"`
echo kill -9 "$pidlist"
for pid in $pidlist
do
kill -9 $pid
done
ipcrm -Q 101
rm "${dnet_home}pidlist"
```

NAME

strpvc.com - start local DNET node PVC service on a VAX system

SYNOPSIS

@strpvc

DESCRIPTION

The **strpvc.com** command allows the system administrator on a UNIX host to start-up the local DNET node. ordinarily run via **dnstart.com.**

SEE ALSO

dnstart(8), dnstop(8), dnstop.com(8), strdgms.com, strstat.com

DIAGNOSTICS

The strpvc.com script is listed below:

```
$ ! strpvc - start DNET PVC Service
$
$ cd dnet_home
$ cd [.bin]
$
$ set proc/priv=sysnam
$ write sys$output "starting dmsdec ..."
$ run/proc=dmsdec dmsdec
$ write sys$output "starting dmstcp ..."
$ run/proc=dmstcp dmstcp
$
$ cd dnet_home
$
```

NAME

strstat.com - start local DNET network status server (on VAX systems)

SYNOPSIS

@strstat

DESCRIPTION

The **strstat.com** command allows the system administrator on a UNIX host to start-up the local DNET node. Ordinarily run by **dnstart.com.**

SEE ALSO

dnstart(8), dnstop(8), dnstop.com(8), strdgms.com, strsta..com

DIAGNOSTICS

The strstat.com script is listed below:

```
$ ! strstat - start DNET status daemon
$
$ cd dnet_home
$ cd [.bin]
$
$ write sys$output "starting dnstatd ..."
$ run/proc=dnstatd dnstatd
$
$ cd dnet_home
$
```

DNET

TECHNICAL GUIDE

Version: 1.18
Print Date: 09/01/89 13:54:37
Module Name: tech.gui

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

CONTENTS

1. DNET Overview 1
   1.1 Underlying Assumptions 1
   1.2 Basic Design Philosophy 1
   1.3 Major Elements of a DNET Network 2
      1.3.1 Network Arrangement 2
      1.3.2 Existing Networks 3
      1.3.3 DNET Hosts 3
      1.3.4 Gateways 5
   1.4 Layered Model for DNET 6
   1.5 Layered Model for Communication Services 8
      1.5.1 Application 8
      1.5.2 Presentation 8
      1.5.3 Session 8
      1.5.4 Transport 8
      1.5.5 Network 9
      1.5.6 Link - Interface 9
      1.5.7 Link 9
      1.5.8 Physical 9

2. Relationships between DNET Components 10
   2.1 Basic I/O Function Library 10
   2.2 DNET Objects 11

3. DNET Permanent Virtual Circuit (PVC) Internals 12
   3.1 Connection Establishment 12
      3.1.1 Summary of Connection Establishment Sequence 13
   3.2 PVC Client Details 17
      3.2.1 Connection Establishment 18
      3.2.2 Close Connection 18
   3.3 PVC - Server 18
      3.3.1 Receive a Connection 18
      3.3.2 Notify Master Server of Session Completion 19
   3.4 Data Streaming During Session - Clients and Servers 20
   3.5 Master Server 20
      3.5.1 Master Server Schematic 21
      3.5.2 Master Server Control Function 22
      3.5.3 Initialization of the Master Server 22
      3.5.4 Example of Application Server Spawning 23
   3.6 Details of Specific Application Server Assignment 23
      3.6.1 Service Assignment Function 23
      3.6.2 Specific Server Instance Table 24
   3.7 DNET Gateways 24
      3.7.1 Permanent Virtual Circuit Relays 26
      3.7.2 Master Server Control of PVC Relays 26
      3.7.3 Detail of PVC Relay Function 26

4. Connectionless Mode Services 28
   4.1 Introduction to Connectionless Service 28
      4.1.1 Schematic of Connectionless Communications Service 28
      4.1.2 Connectionless Datagram Formats 29
   4.2 per protocol DataGram Server (DGS) 31
   4.3 DataGram Master Server (DGMS) 32
      4.3.1 The Routing Function 32
      4.3.2 The Multiplexor Function 33

4.3.3 The DGMS Service Routines 34
4.3.4 The ADGUT 38
4.4 The Connectionless Services Library 40
4.4.1 The Function Of dn_cinit 40
4.4.2 The Function of dn_cwrite 40
4.4.3 The Function of dn_chandler 40
4.4.4 The Function of dn_cread 41
4.4.5 The Function of dn_cdone 41
4.4.6 The Function of dn_salloc 41
4.4.7 The Function of dn_cerror 42
4.5 Component Interaction Diagrams 43
5. DNET Interprocess Communication (IPC) 63
5.1 Introduction 63
5.2 Interface 64
5.2.1 Administration Of IPC Medium 64
5.2.2 Administration Of Individual IPC Mechanisms 66
5.2.3 Sending And Receiving Messages 68
5.3 Implementation 70
5.3.1 The ipcid Table 70
5.3.2 System V 70
5.3.3 BSD 71
5.3.4 VMS 72
6. Miscellaneous DNET Internal Utilities 74
6.1 General System Utilities 74
6.1.1 getppid 74
6.1.2 fperror 74
6.1.3 iosync 74
6.1.4 is_error 74
6.1.5 prttime 74
6.1.6 stricmp 74
6.2 General Network Utilites 74
6.2.1 check_mynet 74
6.2.2 disassemble 74
6.2.3 dn_init 74
6.2.4 dn_makedg 74
6.2.5 dn_makepvc 74
6.3 Stream to Datagram Conversion Utilities 74
6.3.1 strtodg_dglen 74
6.3.2 strtodg_msg 74
6.3.3 strtodg_numhops 74
6.3.4 strtodg_path 74
6.3.5 strtodg_pathlen 74
6.3.6 strtodg_stream 74
6.3.7 strtodg_stream_msg 74
6.3.8 strtodg_type 74
6.4 UNIX Specific Utilities 74
6.4.1 build_argarr 74
6.4.2 execshell 75
6.4.3 startserver 75
6.5 VMS Specific Utilities 75
6.5.1 create_mailbox 75
6.5.2 execshell 75
6.5.3 getargs 75
6.5.4 gobetween 75

6.5.5 setargs 75
6.5.6 startserver 75
6.5.7 lib_do_command 75
6.5.8 lib_spawn 75
6.5.9 sys_assign 75
6.5.10 sys_cancel 75
6.5.11 sys_crelnm 75
6.5.12 sys_crelnt 75
6.5.13 sys_crembx 75
6.5.14 sys_creprc 75
6.5.15 sys_dassgn 75
6.5.16 sys_dellnm 75
6.5.17 sys_delmbx 75
6.5.18 sys_getdvi 75
6.5.19 sys_getjpi 75
6.5.20 sys_getmsg 75
6.5.21 sys_hiber 75
6.5.22 sys_qio 75
6.5.23 sys_qiow 75
6.5.24 sys_trnlnm 75
6.5.25 sys_wake 75
6.5.26 vms_fperror 75
6.5.27 vms_perror 75
6.5.28 vms_read 75
6.5.29 vms_write 75
6.6 MS DOS Specific Utilites 75
7. Interfaces to Underlying Networks 76
7.1 Underlying Network Protocols 76
7.2 TCP/IP 76
7.3 TCP/IP Specific Utilities 77
7.3.1 tcp_accept 77
7.3.2 tcp_close 77
7.3.3 tcp_getclient 77
7.3.4 tcp_initperm 77
7.3.5 tcp_inittrans 77
7.3.6 tcp_open 77
7.3.7 tcp_pvcopen 77
7.3.8 tcp_read 77
7.3.9 tcp_write 77
7.4 DECnet 77
7.4.1 _decnet_read 77
7.4.2 decnet_accept 77
7.4.3 decnet_close 77
7.4.4 decnet_errgeneric 77
7.4.5 decnet_errprotocol 77
7.4.6 decnet_getclient 77
7.4.7 decnet_initperm 77
7.4.8 decnet_inittrans 77
7.4.9 decnet_open 77
7.4.10 decnet_pvcopen 77
7.4.11 decnet_read 77
7.4.12 decnet_select 77
7.4.13 decnet_write 78
7.4.14 vms_aread 78

7.4.15 vms_awrite 78
7.4.16 vms_wait 78

8. User Application Internals
8.1 File Transfer Protocol . . . 79
8.2 Schematic of File Transfer . . . 79
8.2.1 General Considerations 79
8.2.2 ASCII 79
8.2.3 Binary Files 80
8.3 Security During File Transfer
8.4 Initiation of File Transfer from One Remote Node to Another . . . 80
8.5 Initiation of Remote Procedure Upon Completion of File Transfer . . . 80
8.6 Remote Login . . . 81
8.7 Electronic Mail . . . 81
8.8 General . . . 81
8.9 Mail Operation . . . 81
8.9.1 Structure of DENT mail files 82
8.9.2 Sending Mail 82
8.9.3 Reading Mail 82
8.9.4 Mail Routing 82

9. dnetstat - Network Status Function . . . 83

10. Network Command Execution & Task Redirection . . . 84
10.1 Network Command Processor Schematic . . . 84
10.2 Network Command Language . . . 85
10.2.1 Command Language Syntax 85
10.2.2 Using The Command Language 85
10.3 Network Command Interpreter . . . 86
10.3.1 Schematic of Network Command Interpreter 86
10.4 Network Command Server . . . 87
10.4.1 Operations at Network Command Server during File I/O 88
10.4.2 Status Reporting (from last Network Command Server) 88
10.5 An Example . . . 89
10.6 An Example of Network Command Execution . . . 90
10.7 Network Command Processor Implementation . . . 90
10.8 Network Command Interpreter . . . 91
10.8.1 Additional Processing 91
10.9 Network Command Server . . . 92
10.9.1 Implementation of the Network Command Server 92
10.10 Network File I/O . . . 92

11. Presentation Layer Services . . . 94
11.1 XDR . . . 94

12. DNET Error Handling . . . 95

13. Routing . . . 96
13.0.1 get_path 96
13.0.2 load_my_name 96
13.0.3 load_net_table 96
13.1 Router Operation . . . 96
13.2 Routing Example . . . 98
13.3 Routing Table Updates . . . 100

LIST OF FIGURES

Figure 1. DGMS Active Datagram User Table 38
Figure 2. Schematic Overview of Connectionless Service 44
Figure 3. Empty Host Machine With No DNET Components 45
Figure 4. Datagram Master Server Started 47
Figure 5. Process Communication Medium Preparation 49
Figure 6. Invocation of DGS Components 51
Figure 7. dn_cinit process 53
Figure 8. dn_cwrite Process 55
Figure 9. dn_chandler Process 57
Figure 10. Receive Datagram at Destination 59
Figure 11. Receive Datagram: at Gateway 61

1. DNET Overview

This section provides an introduction to the internal functions of DNET. The various elements which make up DNET are described as are some of the important assumptions made in the design. Each of these elements is discussed in more detail in a subsequent chapter.

1.1 Underlying Assumptions

1. Existing (networking) protocols to be employed as the combined link/physical layer for the Heterogeneous Net.
2. DNET Software (operating at transport & network layers) to be resident on all nodes which constitute the Heterogeneous Net.
3. No modification of Operating System Kernels to be required in the implementation/use of DNET Software
4. Initial Operating Environment
   - TCP/IP
   - DECNET

1.2 Basic Design Philosophy

DNET communications and computing services are provided in an environment based on clients and servers. Using the communications services administrative processes request the initiation of server processes; clients then request connections to the servers and obtain the services. Interactive and batch mode operations are supported. The interface to the communications services is provided by a set of input/output subroutines that are included in the user's run-time utility library.

Administrators on each host can regulate the number and type of servers. At connection establishment time the assignment of specific server processes to clients is done using the task initiation facilities of the local operating system. Servers status may be monitored on demand by a network status utility.

In an environment comprising networks with heterogeneous communications protocols, gateways are needed to permit data to pass from one network to another which uses a different communications protocol. Gateways allow programs operating on hosts on different network to communicate with others without concern about possible network protocol differences. Using end-to-end acknowlegements for reliability and automatic inclusion of gateways for protocol conversion DNET provides protocol transparent, reliable, data streams or datagram transmission between hosts on connected DECnet, TCP/IP, (and Asynchronous) networks.

The DNET user can choose to establish a "permanent virtual circuit". In this mode an "open" function is called to establish a communications path from one process to another process in another host, possibly in another network. The path established comprises relay processes and network connections dedicated exclusively to the stream mode transport of data between the end points of the circuit. Permanent virtual circuits reduce the number of network connections that must be established and the associated task initiation required. This significantly improves network performance. When data is transmitted in a "streaming" fashion in one session the performance increase more than offsets the initial cost of circuit establishment.

DNET also provides variable length datagram service. The user interface to this service is connectionless (i.e. no "open" is required before starting process to process communications) however certain local registration as a datagram user is required of processes wishing to use this service. Datagrams may be used to either transmit data or signal information.

1.3 Major Elements of a DNET Network

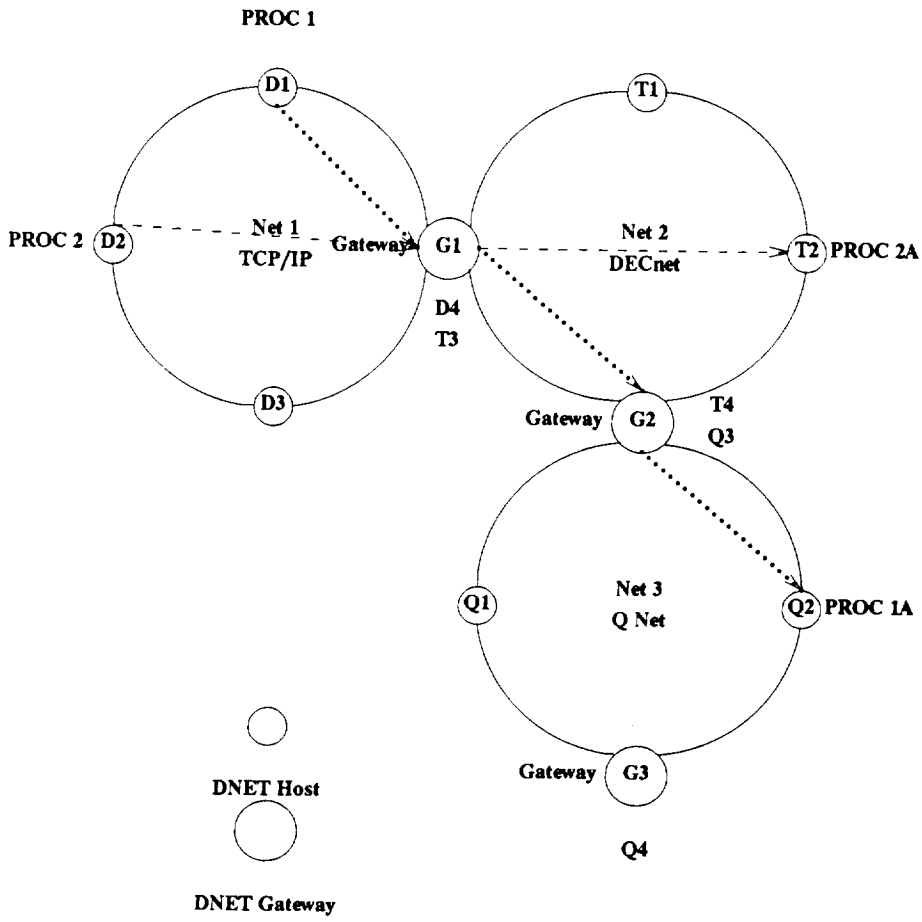
DNET consists of the following major elements:

1. A collection of two or more existing, specific networks
2. DNET Hosts - machines which are able to communicate using DNET services
3. DNET Gateways - special DNET Hosts which also provide protocol conversion between the underlying networks

By implication, the DNET Hosts and Gateways have DNET software installed which establishes their functions. Each of these elements is described in more detail below:

1.3.1 Network Arrangement

DNET is a "meta-network" or a network of networks. The general arrangement of these major elements of a DNET network is shown in the following diagram.



1.3.2 Existing Networks

The underlying networks associated with DNET are ones which have existing reliable, data streaming capabilities. The networks in which DNET may currently operate are TCP/IP and DECnet.

1.3.3 DNET Hosts

DNET Hosts are computers at which local processes may use the facilities of DNET to interact with remote processes in the heterogeneous network. Any computer connected to one of the networks served by DNET may become a node on DNET provided the following conditions apply. The machine must:

1. be resident on a specific existing network (e.g. TCP/IP Net X, SPANET, etc.) which is known to DNET
2. have DNET Host Software Installed & Operational

Each DNET Host contains the following elements:

Software Components

1. **DNET Basic I/O Library** - This is a library of 'C' Language functions which provide the capability to generate, route, read, and write DNET datagrams and to open and close DNET permanent virtual circuits. The major functions of the I/O package are:
   - Establishment of DNET Private Virtual Circuits
   - Send/Receive Connectionless Datagrams & DNET Signals
   - Routing for PVCs and Datagrams
   - Interface to underlying communications network(s)

   These are discussed below.

   1. Private Virtual Circuit Service
      - The DNET user has the option of requesting the establishment of a DNET private virtual circuit (PVC). When a PVC is required the client process makes a call to the DNET function, **dn_open()**, specifying the destination network, host, and process. When a connection to the latter has been established dn_open returns an appropriate channel descriptor to the client which is then used on subsequent **dn_read()** and **dn_write()** function calls.

        Once the DNET PVC is 'open', it appears as a 'smart' wire; i.e. no additional DNET overhead is imposed on the datastream which passes between the communicating processes at each end of the PVC link.
   2. Datagram & Signalling Service

      Two types of datagrams are supported

      1. Connectionless datagrams - these datagrams are used to move user data between remote tasks in connectionless fashion
      2. Signalling datagrams- these are similar to the connectionless datagrams except for their "type" fields and the signal information that they contain in their data fields.
   3. Routing Software
      - DNET employs dynamic, hierarchical routing. Each DNET host maintains a hierarchical routing table in support of this routing function. The paths to hosts in the local network are direct connections. For paths to hosts in other networks, the routing table indicates the host to which connection should be made (or to which a datagram should be sent) next in order to move toward the destination network. The entries in the routing table are currently static, but could be updated dynamically in the future using the general network utility **dnetstat**.
   4. Interface to Underlying Networks connected to the local host
      - Underlying Network Protocols supported in the initial version of DNET include. Interfaces to these networks are established as configuration paramters at the time DNET is installed on a particular host system.
        1. TCP/IP
        2. DECnet
2. **DNET Master PVC Server(s)** - This server process responds to requests from DNET Clients for connection to DNET Application Servers

3. **DNET Application Clients** - These are specific DNET applications such as File Transfer, etc. which may be invoked by the user at each DNET host.

4. **DNET Network Command Interpreter** - operates as special command line interpreter to parse and distribute DNET commands; The latter allow such operations as I/O redirection and distributed command chaining across DNET.

5. **DNET Application Servers** - These are specific servers which are required by a wide range of Network user applications on a continuing basis. The number and types of such servers available at a specific node may vary according to local conditions. Application servers are controlled by systems administrators on hosts in the local domain (network).

6. **DNET Network Command Server** - this 'special' application server interprets DNET Network Command Language commands, executing the local portions thereof, and forwarding those portions of the command to be executed at other DNET hosts.

7. **DNET Datagram Master Server (DGMS)** - this is an internal server process which provides local control for the datagram service and routing for datagrams to remote nodes. All processes which wish to use the datagram service must register with the DGMS.

8. **DNET per Protocol Datagram Server(s)** - these are well-known DNET servers whose purpose is to forward connectionless DNET datagrams to destinations elsewhere in DNET via specific datagram protocols.

Tables and Variables

1. **DNET_Hostname** tbls.myname - Variable containing name of local node and its underlying DNET networks

2. **DNET Routing Tables** - tbls.net. A hierachical routing table which lists the next hop (via a DNET gateway) to move toward all distant DNET networks.

3. **Master Server Init Table** - tbls.msinittcp and tbls.msinitdec. This is a file containing the initialization information for the Master Server. It is loaded into the Master Server Table when the DNET software is started on the local node.

4. **DNET Master Server Table** - This table contains a list of allocated DNET application servers on this host and their status (not-running, idle, in-use). It is used by the Master Server in responding to DNET clients' requests for service.

5. **DNET Server Instance Table(s)** - These tables list detailed instances of Specific DNET servers under control of the Master Server at this DNET host. There is a separate SIT for each type of server available at this node.

6. **Connection Lock Table** - (not implemented at this time) Used by the DNET Datagram Service; lists process/channel/streams currently connected to this host which may be used for the forwarding of connectionless datagrams

1.3.4 Gateways

DNET Gateways are nodes in DNET which are connected to one or more networks in which DNET is operating. The function of the gateway is to bridge the protocol and other differences between these networks in a transparent manner. The gateway functions are implemented in special DNET **PVC Relay** servers and **Datagram Servers** which provide protocol conversion for Permanent Virtual Circuits and connectionless datagrams respectively.

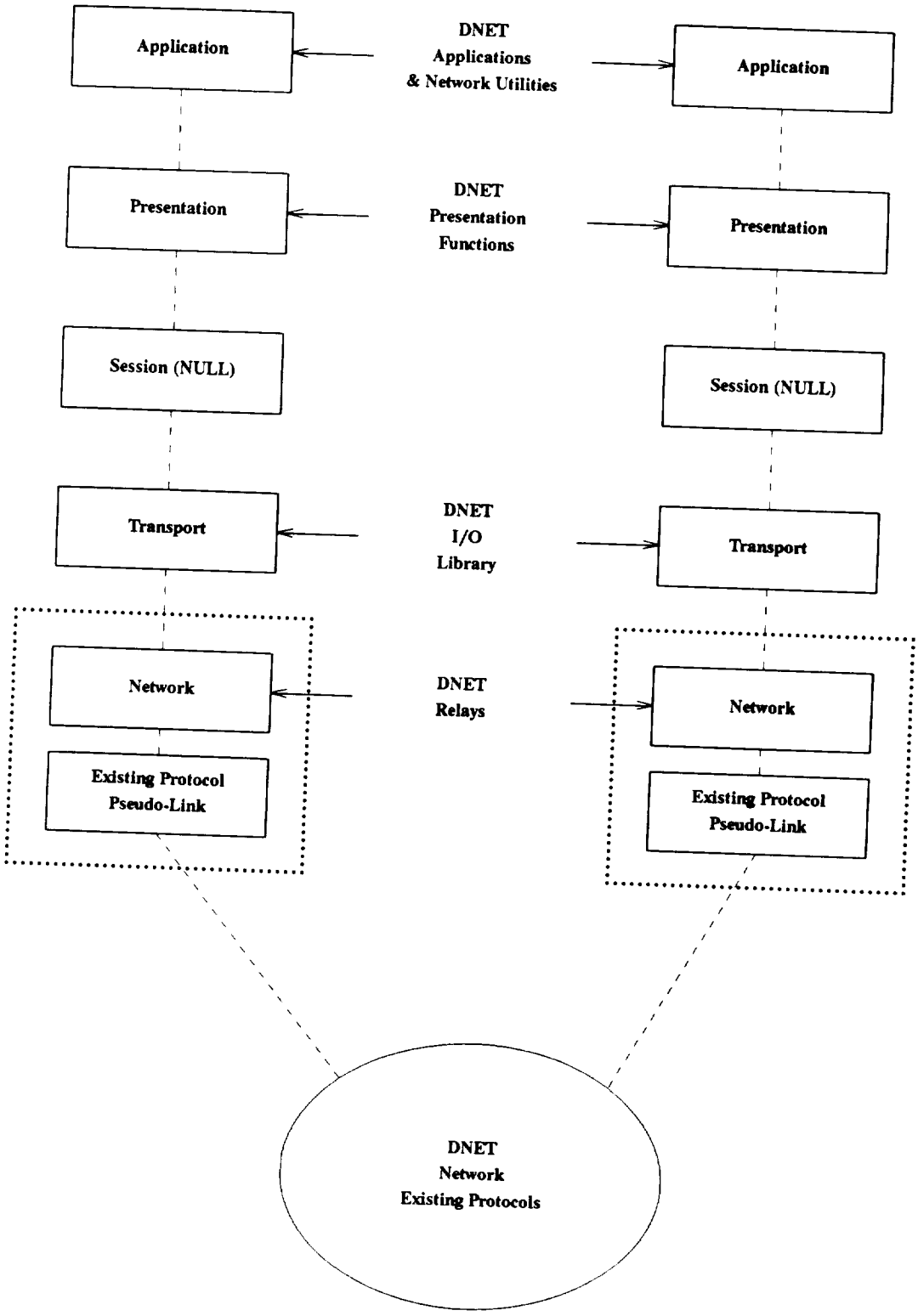
Specific Gateway Elements include all elements of other DNET hosts with the following variations and additions:

Software Components - All of DNET hosts plus

1. **PVC Relay Server** - these special processes perform protocol hops for PVC client/server conversations between two different underlying networks forming a portion of DNET. When idle, these relay processes are like DNET application servers. The PVC Master controls allocation of the relays when they are required to satisfy a connection request from a client. Once a PVC has been established, the relay performs efficient, full duplex streaming between the client and server processes.

Tables - Same as (non-gateway) DNET hosts

1.4 Layered Model for DNET


Application
DNET
Applications
& Network Utilities
Application
Presentation
DNET
Presentation
Functions
Presentation
Session (NULL)
Session (NULL)
Transport
DNET
I/O
Library
Transport
Network
DNET
Relays
Network
Existing Protocol
Pseudo-Link
Existing Protocol
Pseudo-Link
DNET
Network
Existing Protocols

1.5 Layered Model for Communication Services

The following is a introduction to the services provided at the various layers within DNET Software. So far as deemed practical, the model attempts to be consistent with services defined by the ISO-OSI Model. No claim is made that this consistency is exhaustive, however. Rather, the use of terminology, concepts, and naming conventions, from the OSI model are intended to allow a more detailed future migration that model's environment.

1.5.1 Application

Provides library of function calls which DAVID (or other applications) may invoke in order to converse with remote nodes on the heterogeneous net

Application Services Supported:

1. File Transfer
   - ASCII and Binary
   - End to End Acknowledgement
   - Data Structures mapped end to end if context registered with Presentation layer Service
2. Remote Login
3. Remote Execution
4. Mail
5. General Utilities
   - Status of all network nodes (up/down)
   - Load on remote node
   - Hostid, hostname, alias resolution

1.5.2 Presentation

- The SUN External Data Representation (XDR) Specification is used to allow machine independent sharing of data types across all DNET nodes.

1.5.3 Session

- This layer is null at present; All connections are assumed to support only one simultaneous session

1.5.4 Transport

- DNET Basic I/O Function Library
- Reliable Task-to-Task Communications
- End-to-End Acknowledgement of Files, etc.
- User Authorization, Access Privileges

1.5.5 Network

- Defines routing strategies on the Heterogeneous Net
- Provides Relay function at intermediate Nodes
- Self Contained within DNET I/O Library, PVC Relays, and Datagram Servers

1.5.6 Link - Interface

- A Pseudo-Link facility which provides consistent interface to a variety of underlying network protocols
- Generally, the calls from the network layer to this interface map on a one-to-one basis to calls in the underlying, well known protocol.

1.5.7 Link

- These layers provided by underlying protocols

1.5.8 Physical

- Data is assumed to move in a reliable, streaming fashion on any of these links

2. Relationships between DNET Components

2.1 Basic I/O Function Library

The function calls provided in the DNET basic I/O library are summarized in the following table:

Generic Operation	VIRTUAL CKT Client	VIRTUAL CKT Server	Datagram	SIGNAL
Estab. Connect.	dn_open	dn_getclient		
Write	dn_write		dn_cwrite	dn_signal
Synch Read	dn_read		dn_cread	Dest Oper Sys
ASynch Read			dn_cdg_handler	Dest Oper Sys
End Connect.	dn_close	dn_done,dn_close	dn_cdone	

These functions are described in the following sections according to the type of service (PVC, Connectionless Datagram, or Signal) which they support.

2.2 DNET Objects

The following table shows the relationships between the various DNET components

DNET Object Table

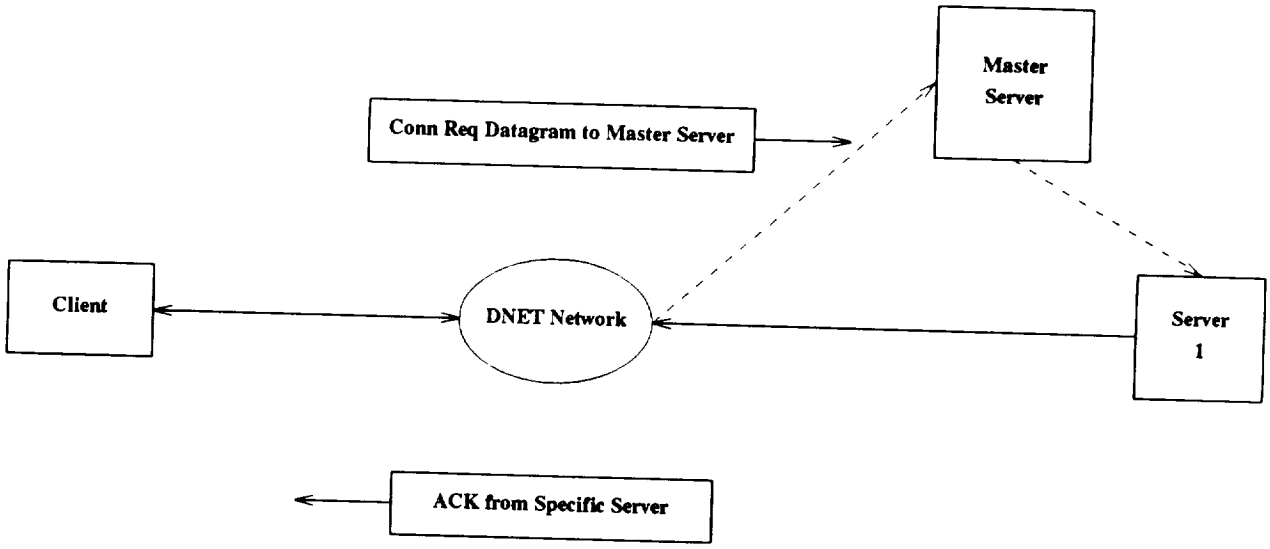
Object Name	Resides in	Started by	Started at	Updated by	Receiver	Monitored by	Shutdown by	Shutdown at
generic applications servers FXFR, RLOGIN, REXEC ,etc.	various hosts	server control function	some at start up,others as needed	-	server names sent to requestors by server assignment	Master Server Process	Master Server Process	N/W shutdown or reduced need
Connection Lock Table	Each DNET Host	DG Server	when perm connection	DG Server	DG Server	-	-	-
DNET Basic I/O pkg	application program	user program	proc start	-	-	part of application program	end of proc	end of proc
Master Server Table	host where M S is	Mast Serv	start of M S	M S	M S	M S proc	net shutdn	net shutdn
Master Server	Each DNET host	start of DNET S/W	-	-	-	net shutdn	net shutdn	
Gateway	Hosts w 2 or more N/Ws	server cont fcn in DS	system start	-	client calls	DS proc	Sys Admin	net shutdn
N/W Comm Lang Interprtr	Each DNET host	User	as needed	-	-	N/W Util	User	
when complete								
N/W Command Server	Each DNET host	Master Server	Comm Exec	-	-	M S Proc	M S Control	when complete
Local Routing Table	all DNET hosts	local Sys Admin	net Start	N/W Status Client	Net Status Server at G/W	-	-	-
N/W Stat Server	Gateways	any i/o	runs period	-	reports to Net Stat Cl	Net Stat Cl	end of appl prog	proc term
Service Assign Fcn	in M S Proc	Part of M S	at net Start	-	Master Server	-	Net Shutdn	Net Shutdn
' S Control function	In M S Proc	Part of M S	at N/W Start	-	controls # servers via M S Tab	Adm Host Net Stat Cl	Net Shutdn	Net Shutdn
Well Known Server Table	N/W Admin Host	N/W Admin Server	N/W Start	N/W Admin Server	sent to all hosts by N/W Admin server	-	-	-
Datagram Server	All DNET Hosts	Loc Sys Admin	DNET Start	-	Remote DG Serv; Local DNET proc	N/W Util	N/W Shutdn	N/W Shutdn
PVC Relay	Gateways	Mast Server	As needed	-	Next Hop	Mast Server	Mast Serv	
when complete								
Server Instance Table	Each DNET Host	Mast Serv	N/W Startup	Mast Serv	-			
Mast Serv	Mast Serv	N/W Shutdn						

3. DNET Permanent Virtual Circuit (PVC) Internals

The section describes the several function calls associated with DNET Permanent Virtual Circuits. The functions are arranged to indicate those used by DNET client and server processes.

3.1 Connection Establishment

This section provides additional details on the DNET PVC connection establishment operations. The basic client-server connection establishment procedure is shown in the following diagram:



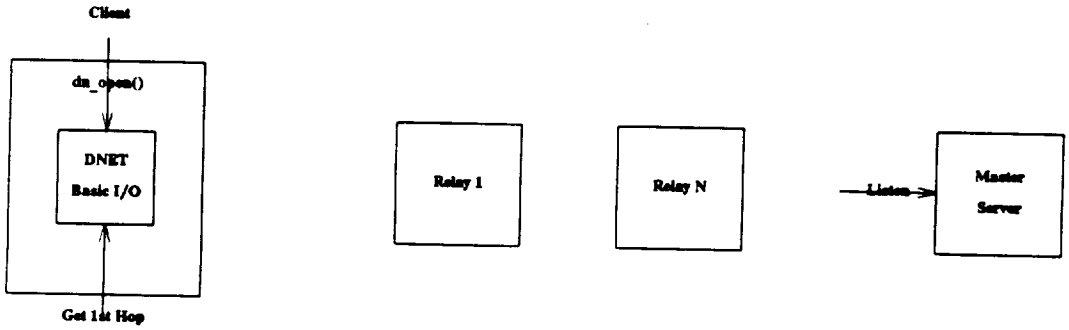
1. The client calls dn_open() with parameters including:
   - Network Name
   - Host Name
   - Name of Server
2. **dn_open()** obtains a network address for itself from the local network software. This address is sent as part of the connection request datagram to the Master Server.
3. If relays are used (required), the relay processes recognize the connection request datagram and do not close the connections following transmission of the datagram.
4. The last relay (or the Basic I/O package on the client if there are no relays) connects to the Master Server at the destination host; it places its network address in the connection request datagram as the "call back" address. The network address of the client is preserved for possible use by the server.

5. The Connection request datagram is delivered to the Master Server.
6. Using the Master Server Table and a Specific Server Instance Table, the Master Server allocates/spawns (VMS/UNIX respectively) a particular instance of the requested server type.
7. If service cannot be provided by the Master Server, a service_denied or "NAK" response is returned to the client.
8. If a specific server can be provided, the Master Server passes the Connection Request Datagram to this server, sends an "ACK" to the client, then closes its connection to the preceding process in the connection chain.
9. The specific instance of the server calls **dn_getclient()**. Depending on the state of the "callback_flag" in the Connection Request Datagram, dn_getclient() performs either a call_forward or call_back procedure to complete the connection.

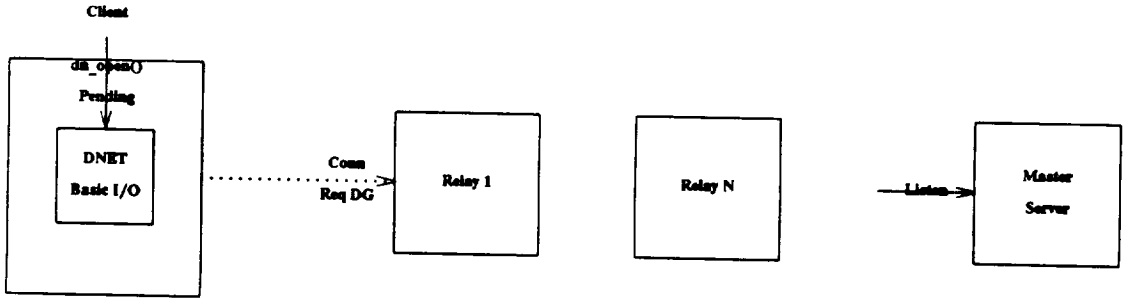
NOTE: If the user wishes to use a specific **user-defined** process (not a known DNET service) that process name should be specified in the initial call to dn_open(). **dn_open()**, using the networking software of the local system, spawns a copy of the named process if that process does not exist already.

3.1.1 Summary of Connection Establishment Sequence

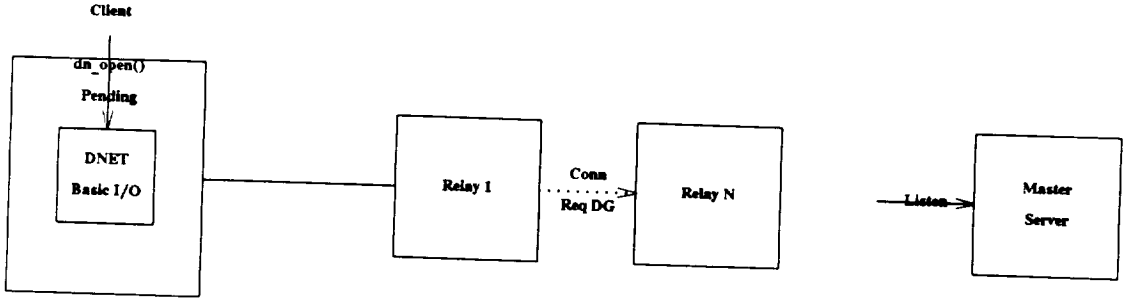
The several operations described above are shown schematically in the following series of diagrams:



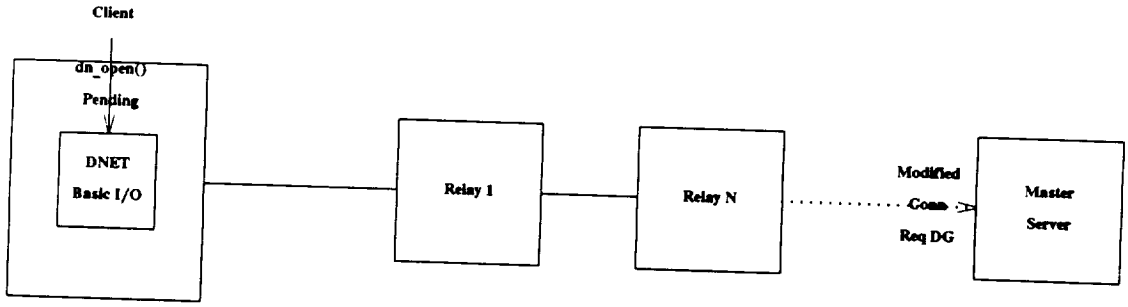
Client Calls dn_open()



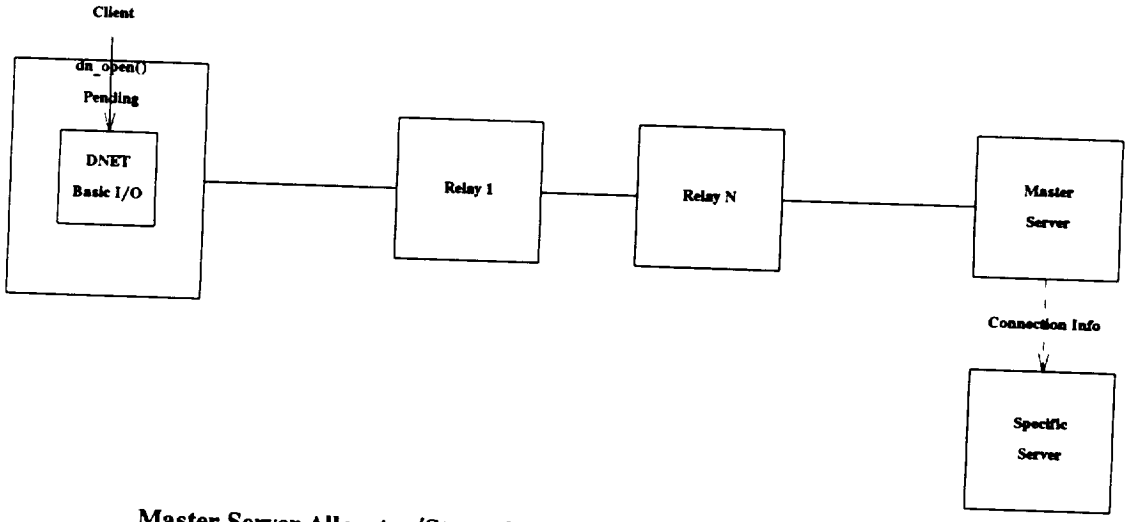
Connection Request Datagram Issued



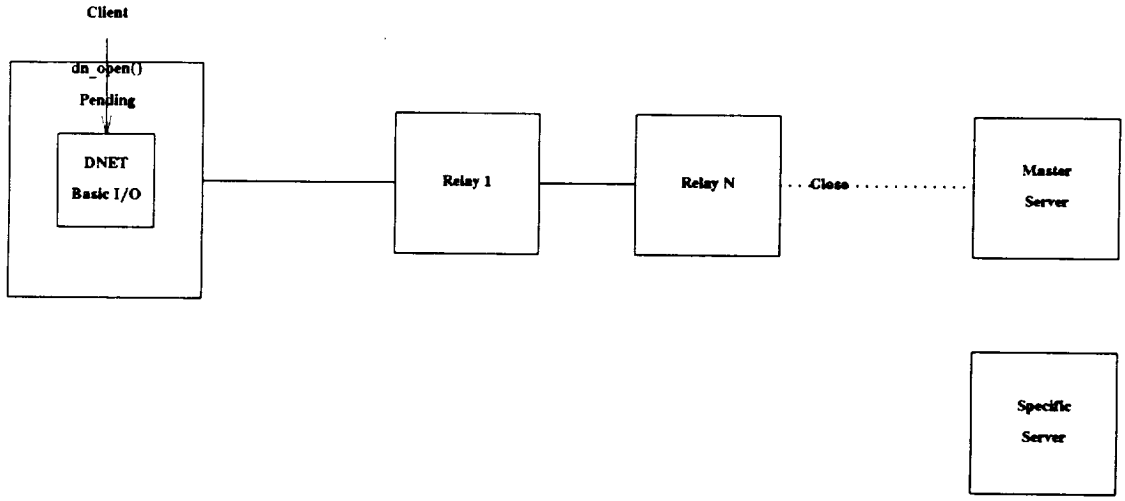
Connection Request "pulls" Permanent Virtual Circuit Open



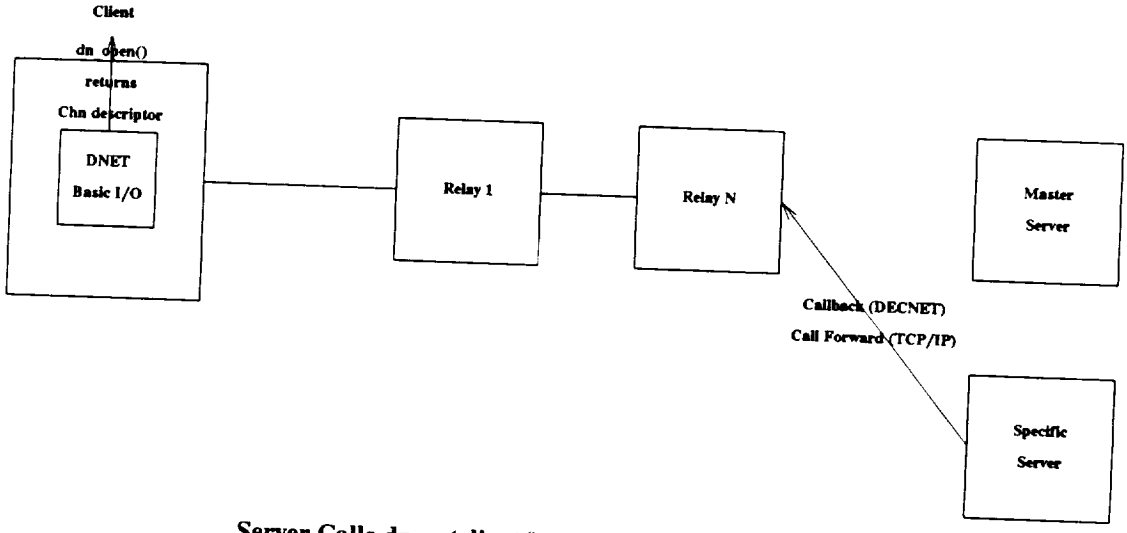
Last Relay Sends Final Connection Information to Master Server



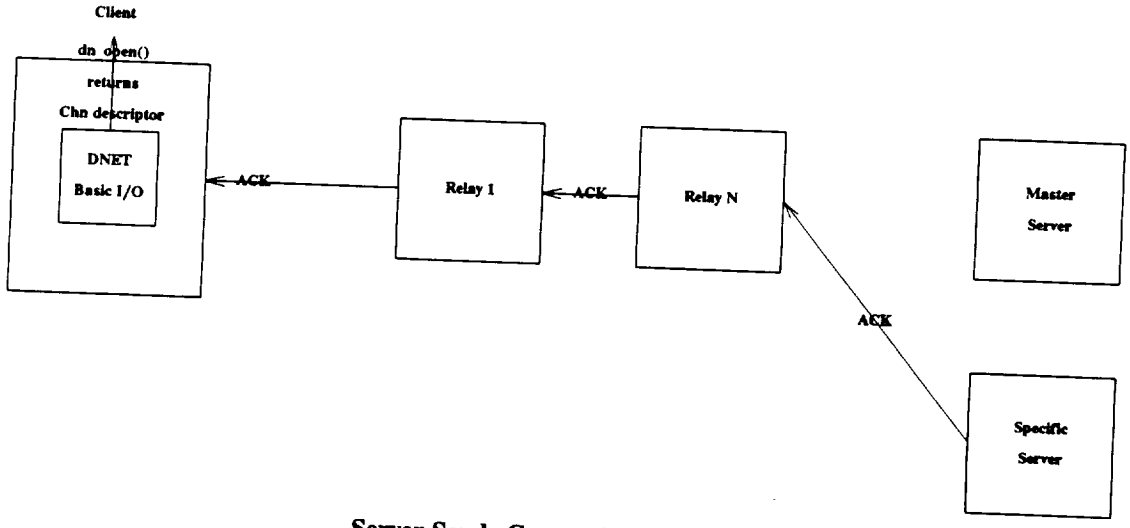
Master Server Allocates/Starts Specific Server & 'hands off' Connection Info



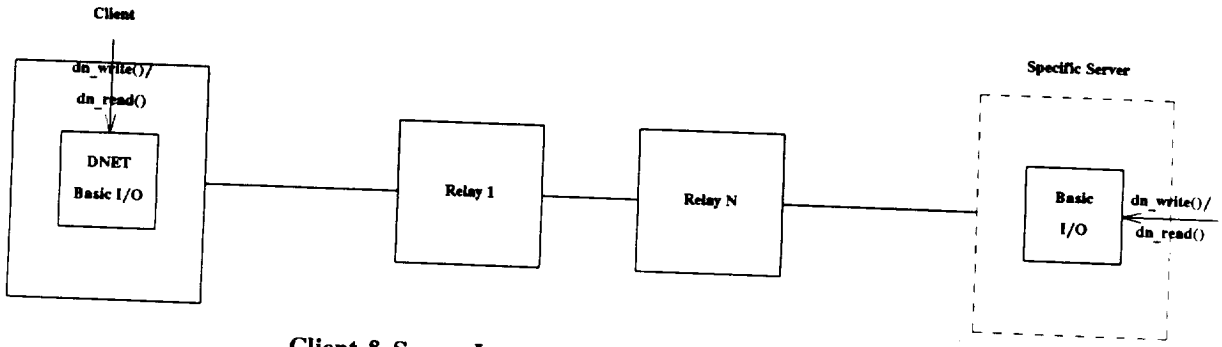
Master Server Closes Connection to Last Relay



Server Calls dn_getclient() to complete connection to Client



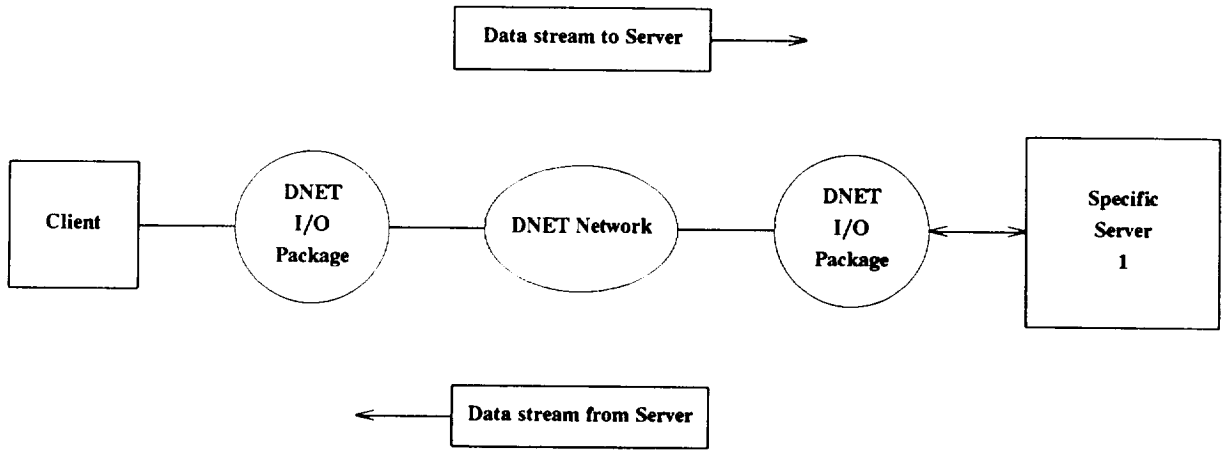
Server Sends Connection ACK to Client



Client & Server Interact via dn_write() & dn_read()

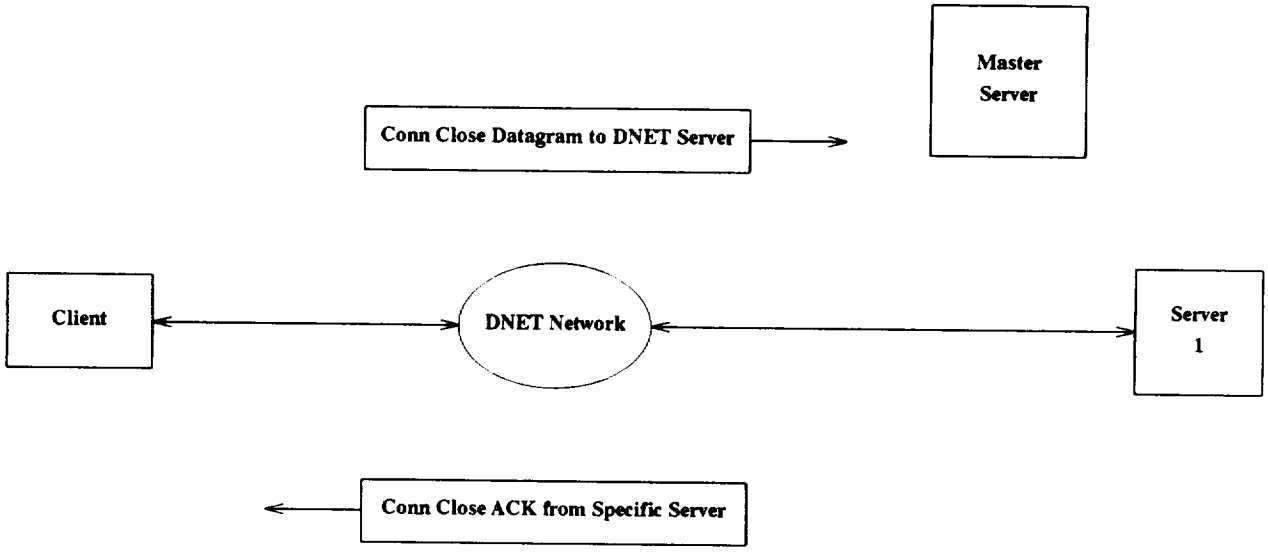
3.1.1.1 Client Server Conversation

Once the PVC is 'open' data is streamed between client and server processes:



3.1.1.2 Closing a Client Server Conversation

At the conclusion of a session, the DNET permanent virtual circuit may be closed by calling **dn_close()**.



3.2 PVC Client Details

A DNET Client Process employs the following calls for Virtual Circuit Service:

3.2.1 Connection Establishment

dn_open

chan = dn_open(net, host, service)

```
int chan;       /* A channel number;
                   used in subsequent read and write calls */
char *net;      /* A DNET network name */
char *host;     /* A DNET host name */
char *service;  /* A  DNET service */

char *userid;
char *passwd;
```

dn_open() is used by client processes to request a Private Virtual Circuit connection to the specified service a given network and host. The function does not return until a path to the destination has been opened or an error conditions occurs.

3.2.2 Close Connection

dn_close

```
status = dn_close(stream)
    int status;     /* An indication of success or failure */
    int chan;   /* A channel structure that was
                   previously opened using dn_open() */
```

dn_close() closes a communications channel; it can be used in clients and servers.

3.3 PVC - Server

3.3.1 Receive a Connection

dn_getclient

chan = dn_getclient(service, usrbuf, pusrbuflen)

```
char* service;
char* usrbuf;
char* pusrbuflen;
```

dn_getclient is invoked by all DNET application servers in order to establish connections with clients which request this service.

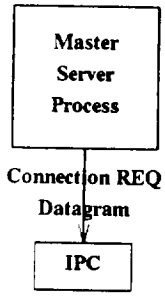
Functionally, dn_getclient() waits for the Master Server to 'hand' it a service request (in the form of a valid 'channel' or stream descriptor) for the current application server.

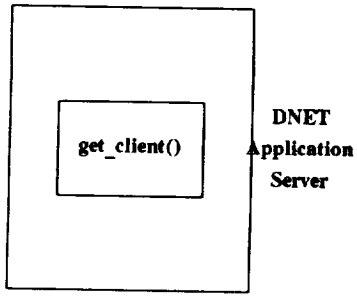
Internally, dn_getclient() is slightly more complex:

1. A DNET IPC is created for 'receipt' of future service requests.
2. **dn_done()** is called to 'register' this instance of the server as available in the appropriate Server Instance Table.
3. The program then 'blocks' on **ipcrcv**; it is waiting for an actual service request to arrive from the Master Server
4. when **ipcrcv** returns, its contents are inspected and disassembled using the function **disassemble.**
5. If the Datagram type is **callback,** dn_open() is used to call back. When dn_open() returns a channel, this descriptor is passed on to the waiting application server.
6. If the CR Datagram type is 'stream', the channel descriptor for this call is passed as part of the datagram. dn_getclient transparently passes this channel descriptor to the server.

   NOTE: the callforward mode is not currently activated at any server The call forward mode is intended for use in Master Server/Application Server relationships where an 'open channel descriptor' may be passed from a parent to a child process. UNIX/TCP/IP supports such channel passing with ease, DECnet does not.

7.





3.3.2 Notify Master Server of Session Completion

dn_done

dn_done is called by each DNET Application Server before exit to indicate to the local Master Server that it has completed its task and is available for use

dn_done is also called (the first time thru) within dn_getclient to register the server as available with the Master Server

dn_done() uses a common IPC (DMS_TCP or DMS_DEC depending on the environment)

3.4 Data Streaming During Session - Clients and Servers

The functions **dn_write() and dn_read()** are used by both clients and servers to 'talk' on an open DNET PVC Stream. These functions are equivalent to the UNIX system calls **write() and read()**; the chan on which the operations occurs is an open DNET channel.

dn_write

nbytes = dn_write(chan,buf,nbytes)

```
int nbytes;     /* The number of bytes, including DNET headers,
                   that was written on the given stream. */
int chan;       /*  I/O channel returned from dn_open  */
char *buf;      /* The data that is to be sent. This function
                   prepends the data with a DNET header. */
```

dn_write() takes data and packages it in a datagram for transmission over the appropriate communications channel.

dn_read

Synchronous (Blocking) read

```
nbytes = dn_read(chan, buf, count)
        int nbytes;     /* The number of bytes, including DNET headers,
                        that was read from the given stream. */
        int chan;       /* A pointer to an I/O structure that was
                           previously opened by dn_open() */
        char *buf;      /* A result parameter where the datagram, in
                           string format, is placed; this buffer
                           contains the DNET headers. */
        int count;      /* The maximum number of bytes to receive. */
```

dn_read() reads a datagram from the communications channel and unpackages it based on its type.

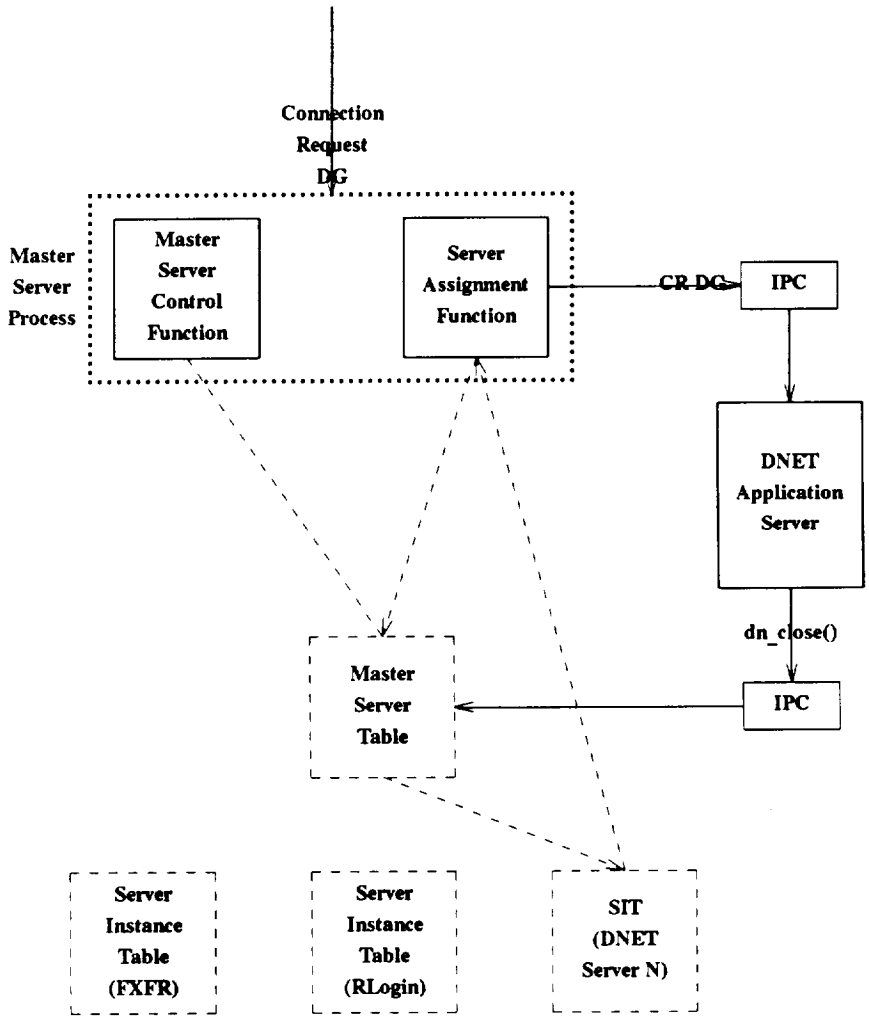
3.5 Master Server

This section describes the operation of DNET Master Servers. Master Servers are used to control the DNET application processes within a single domain (underlying network) on the heterogeneous network. The Master Servers are located at any computer attached to the local network which is to be

considered a DNET Host.

Master Servers are also used at DNET gateways to allocate Permanent Virtual Circuit Relay processes. Since Master Servers 'listen' on only one specific underlying network, DNET Gateways must have a separate Master Server for each network to which it can provide relay services. (See the Chapter on Gateways for additional information).

3.5.1 Master Server Schematic



The Master Server Process has several separate functional elements as indicated in the above figure:

- Server Control Function
- Server Assignment Function

The Master Server utilizes the Master Server Table and Server Instance Tables in providing application services.

3.5.2 Master Server Control Function

This function has responsibility for the allocation and spawning of DNET application servers within the local domain.

Prespawning of processes is available as an option in order to improve the response time of certain systems such as VAX/VMS in which process start up time is signficant.

3.5.2.1 Application Server Spawning Algorithm

1. At network start up spawn a number of copies of the servers, keeping their process id's for later use in forming the process names to give to clients. After giving a server to a client, spawn another to replace it.
2. For less frequently used services- Spawn only when a client requests a server. This is the Transient Server. Generally used in UNIX-TCP/IP cases.
3. Give the same prosess name to every client with the extension '_X' where X is the xth instance of the server, up to the limit specified in the Master Server Init Table . To keep the the client count accurate the server must signal the Master Server by calling **dn_done()** prior to terminating at the completion of a session.
4. For very frequently used services- Spawn the maximum number desired and have server listen for the next client when they complete their service for a client, and at the same time notify the Master Server that they are ready for assignment.

3.5.3 Initialization of the Master Server

3.5.3.1 Master Server Init Table

The Master Server Init Table is read when the Master Server is started at the local DNET host.

DNET Master Server Init Table

Server Type	Image Name	# Prespawned	Max #	Init #
dechod	dechod	1	8	3
dtftpd	dtftpd	1	9	4
drexec	drexec	1	1	1
dnstatd	dnstatd	1	1	1
dncld	dncld1	1	10	5
dlogind	dlogind	1	10	5
dmaild	dmaild	1	10	1

This is a flat ASCII file which may be edited by the local system administrator.

3.5.3.2 DNET Master Server Table

The Master Server Table is a dynamic indicator of the types of DNET application servers available at the local DNET host, the number which are currently available, whether these processed are prespawned, the maximum number available, and the number currently in use, together with pointers to a Server Instance table for each specific server type.

An example of the Master Server Table is shown in the following diagram:

DNET Master Server Table					
ServerType	PreSpawned	Max #	# Avail#	In-Use	Ptr to SIT
drexecd	Y	10	5	2	78555
dtftpd	Y	10	5	2	79747
dnmail	Y	1	1	1	83297
dncl	Y	10	5	2	99541
drelaydt	Y	10	5	2	81423

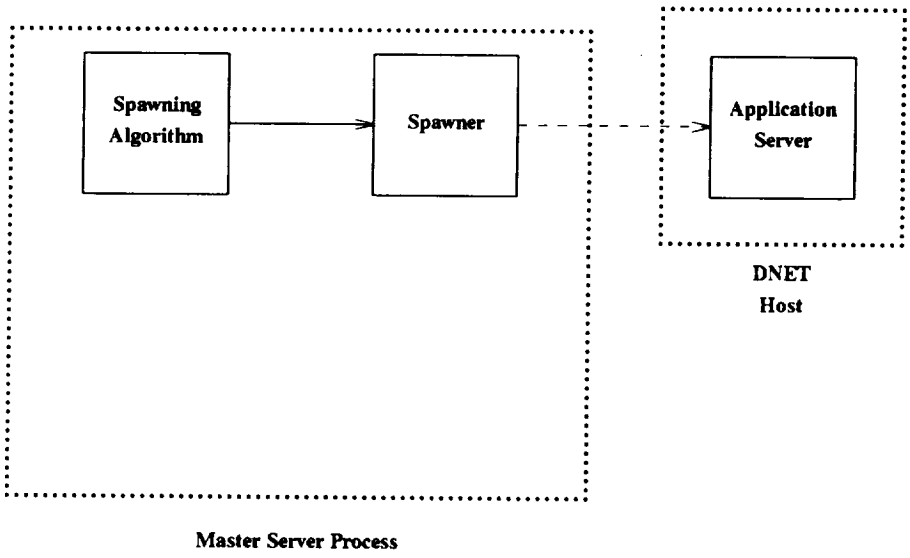
This table is maintained dynamically by the Master Server internal to itself. It may be read using

dnetstat

Use of this function is described in the USER's and ADMINISTRATOR's guides.

3.5.4 Example of Application Server Spawning

The server spawning procedure is shown in the following diagram:



3.6 Details of Specific Application Server Assignment

3.6.1 Service Assignment Function

The Service Assignment function of the Master Server Process has the task of responding to requests for service from application Client Processes at DNET hosts.

1. Accept Application Server Requests as they arrive at Master Server

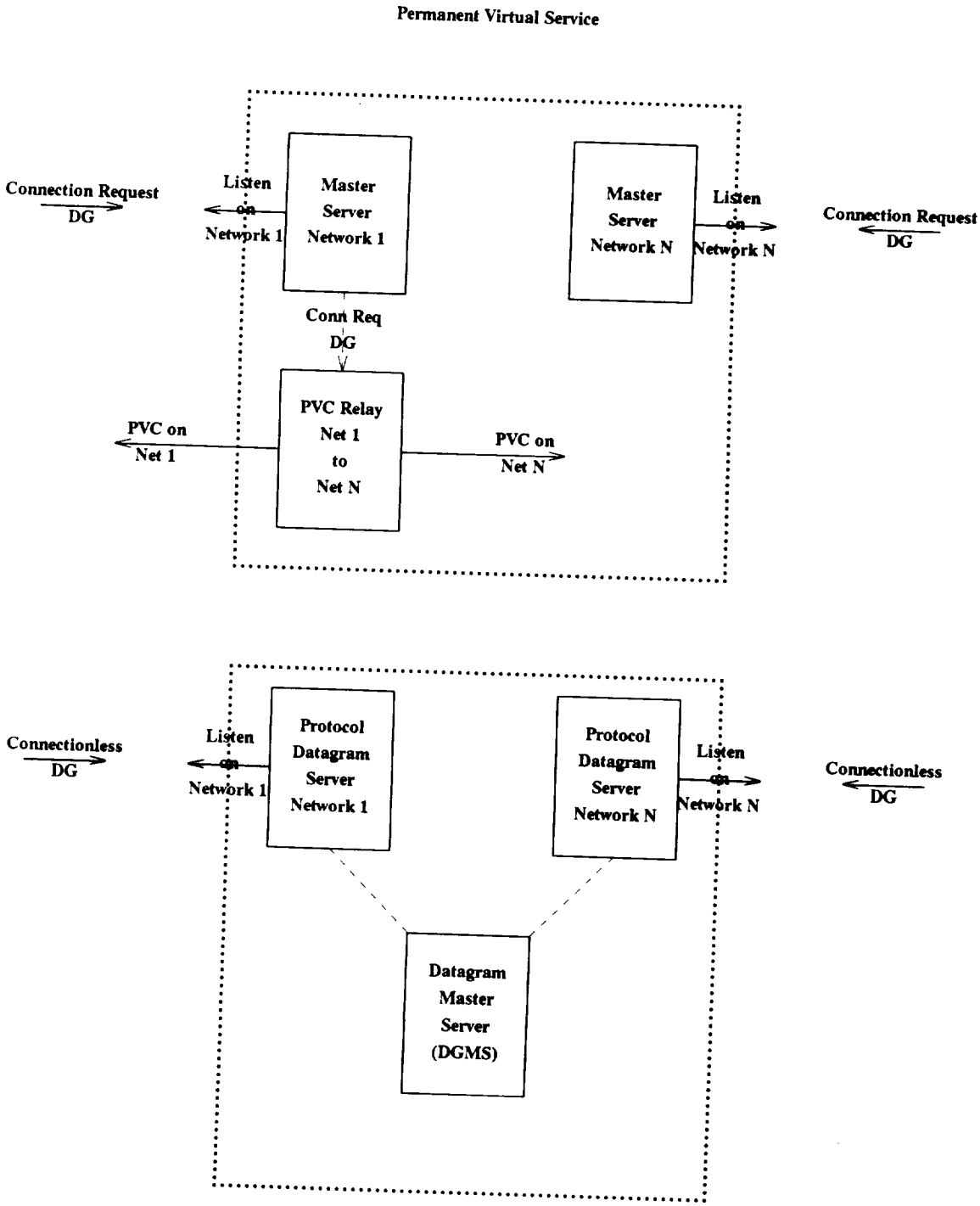
2. Find available Application Server (or spawn one) by examining Master Server Table and Specific Server Instance Table.
3. Send Connection Information to the Specific DNET Server assigned to this request via DNET IPC Mechanism.
4. Flag server as In-use in Master Server Table

3.6.2 Specific Server Instance Table

DNET Server Instance Table				
Server Type = File Transfer				
PID	In-Use	Time Started	Time End	Ptr to MST Entry
1322	Y	10:11	-	45779
1377	Y	10:15	-	45888
1422	N	10:15	10:20	45995
1428	N	10:16	10:21	46100
-	-	-	-	-
-	-	-	-	-

3.7 DNET Gateways

DNET Gateways are nodes in DNET which are connected to one or more networks in which DNET is operating. The function of the gateway is to perform protocol conversion between these networks in a transparent manner. Following terminology used by Space Telescope Institute and others, the protocol conversion are performed via DNET **relay** servers. The relay functions are implemented in special DNET **PVC Relay** servers and **Datagram Servers**. The PVC (Permanent Virtual Circuit) servers support private circuits between communicating tasks while the Datagram Servers perform relay tasks for DNET connectionless datagrams.



DNET Gateway Elements

3.7.1 Permanent Virtual Circuit Relays

The PVC Relay Servers provide a means of moving DNET stream data between two different network protocols. Each relay process is tailored for a specific protocol conversion task. The relays server accepts calls from one protocol/network and then establish a full duplex channel.

PVC Relays are allocated by DNET Master Server Processes in the gateway machine and may thus be considered as special purpose DNET Application Servers whose primary function is protocol conversion on a data stream. Since Master Servers can only listen on a single network, a gateway has a Master Server (and a corresponding 'pool' of relay processes) for each protocol boundry it supports. Each Master Server accepts connection requests from a particular side of a protocol boundary and allocates relays from this pool to service the request.

Relays can be used in routing and communications load balancing. Adding additional relay processes to a gateway reduces the delay in accepting data from the network.

3.7.2 Master Server Control of PVC Relays

PVC Relays are controlled by the DNET Master Server in the Gateway machine. The interactions between the Relays and the Master Server are very similar to those of any DNET application server. Startup and connection passing are identical to other servers. Thus, the relay calls **dn_getclient()** to complete the connection to the preceding element in the connection chain.

The 'application' element of the relay requires establishment of a forward connection on the next hop required as part of the connection establishment. This is accomplished by having the each relay call dn_open() on its 'server' side.

The detailed steps in starting up the PVC relay are as follows:

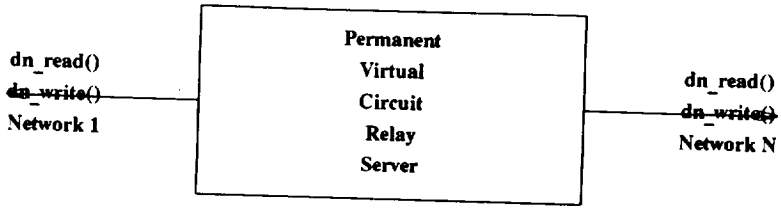
1.
2. Master Server spawns/allocates 'next_service' (the appropriate PVC relay in this case) and hands the entire 'CR DG' to the Relay Server.
3. The allocated Relay Server is waiting on return from **dn_getclient()**
4. Waits for "ACK" to be returned from the 'dest_service' (through its call to dn_getclient). The 'dest_service' would make the last 'dn_getclient' call.

   stream = dn_getclient ( );

   When dn_getclient returns, the relay then calls **dn_open()** passing the CR DG to this function:

3.7.3 Detail of PVC Relay Function

The major elements associated with PVC Relays are shown in the following diagram:

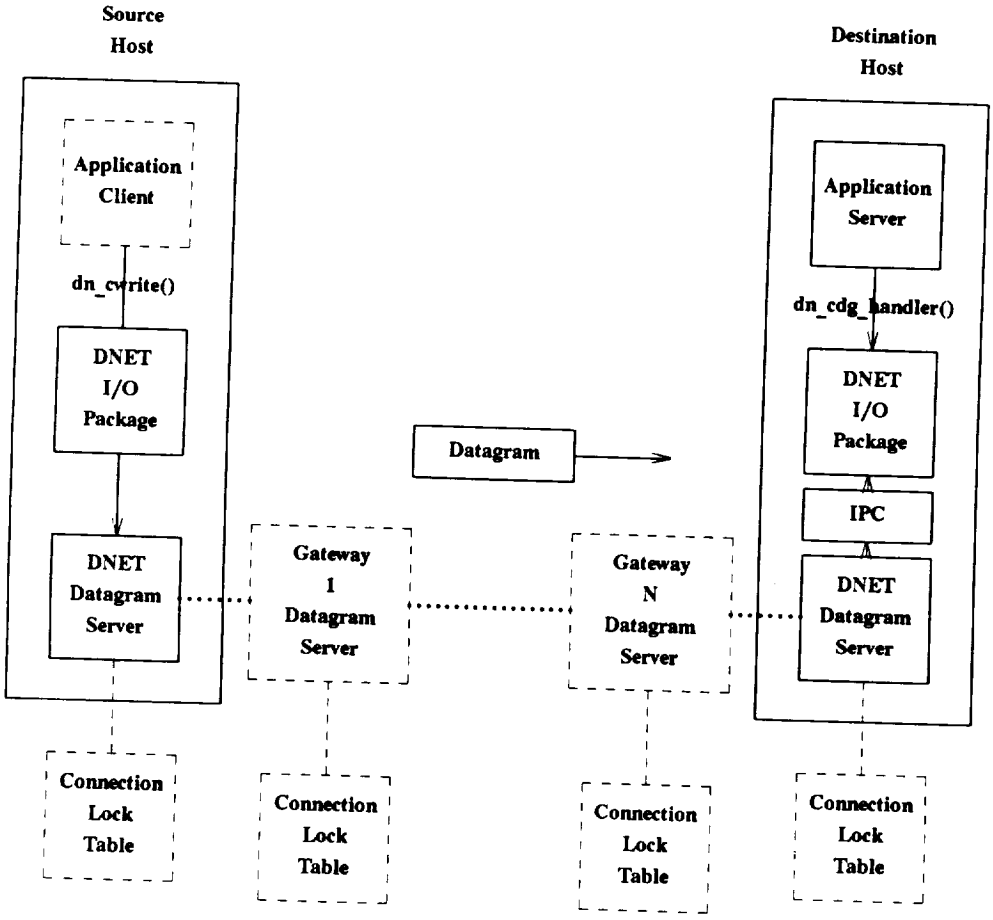

dn_read()
dn_write()
Network 1
Permanent
Virtual
Circuit
Relay
Server
dn_read()
dn_write()
Network N

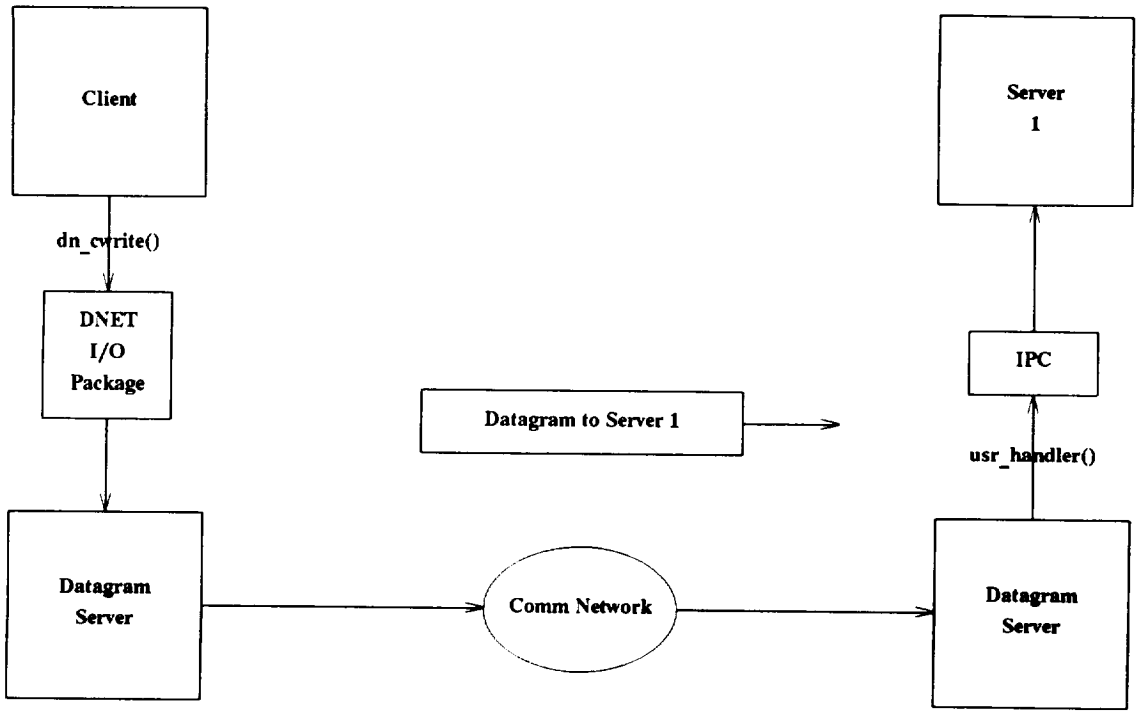
4. Connectionless Mode Services

4.1 Introduction to Connectionless Service

4.1.1 Schematic of Connectionless Communications Service

The connectionless communications service is shown schematically in the following diagram. Client and server process pairs employ the DNET BASIC I/O Library to generate datagrams which are routed automatically via DNET Datagram Servers to the destination process.





Applications using the connectionless mode of this service call only two library functions: **dn_cwrite()** & **dn_cdg_handler()**. The initiating process (process sending the datagram) invokes **dn_cwrite()**. Processes which expect to receive datagrams, (in general, all DNET applications), must call **dn_cdg_handler()** at start up to identify an "asynchronous completion routine" to be executed whenever a connectionless datagram arrives for this process. More complete details on the datagram service are provided in a separate Chapter.

4.1.2 Connectionless Datagram Formats

The general format of a DNET datagram is:

```
struct    ass_dg_buf
{
          char      desthost[I_MAXHNAME]
          char      destnet[I_MAXNNAME]
          char      destproc[I_MAXPNAME]
          char      srchost[I_MAXHNAME]
          char      srcnet[I_MAXNNAME]
          char      srcproc[I_MAXPNAME]
          int       maxhops;
          int       buflen;
          char      buf[D_MAXDG];
};
```

The DNET connectionless services provides a standard connectionless interface to a heterogeneous pool of underlying protocols. The underlying protocols include:

- Operating Systems
  - UNIX System V.2
  - UNIX BSD4.2
  - VMS
- Networks
  - UDP/IP
  - DECnet

The DNET user applications will be provided with connectionless service totally independent of any of the above possible combinations through the following major components:

per protocol DataGram Server(DGS)

The basic function of these components is to provide a standard interface to the DGMS for all underlying protocols. There will be one dgs module set for every underlying network protocol (UDP/IP, DECnet). The UDP dgs module set consists of two modules: one for reading incoming datagrams and one for writing outgoing datagrams. The DECnet dgs module consists of one module which both reads incoming while sending outgoing datagrams.

All dgs modules are written as standard dnet datagram programs. This is to say that the dgs modules interact with the dgms using the same library routines as any other datagram application. The difference being that they have a specific function to perform that is imperative to the operation of dnet. That function is to act as a dumb relay between underlying network provider and the dgms module.

DataGram Master Server (DGMS)

As the name implies, this component coordinates the activities of all DNET connectionless components. The dgms module provides two basic services for the dnet datagram service: routing and multiplexing of datagrams.

The routing procedure is driven by the same routing tables used by the dnet connection services. The dgms, though, is the only component of the dnet connectionless services that provides routing. All other modules know only how to pass a partially qualified datagram off to the dgms. The dgms looks up the destination network in the routing table and uses the next network protocol to determine which dgs module set to pass the datagram on to (It may also just pass it directly to a server process if the datagram is already on the destination machine).

The multiplexing service provided by the dgms is driven off of an internal table (ADGUT - Active Datagram User Table) which has a record for every communication endpoint provided. Included within these records is a string value representing the name that is bound to a given endpoint. This name is used to identify which process is to receive a datagram. All processes must bind to a process name at the time they call dn_cinit if they expect to receive datagrams. The following process names are reserved for the datagram services and should not be used in user applications:

- dgsudp
- dgstcp
- dgsdec

Connectionless Services Library

The connectionless services library will provide DNET connectionless user applications with a variety of standard subroutines to access the connectionless services. These subroutines include:

- dn_cinit()
- dn_cwrite()
- dn_chandler()
- dn_cread()
- dn_cdone()
- dn_salloc()
- dn_cerror()

4.2 per protocol DataGram Server (DGS)

The primary purpose of the DGS module set is to provide a simple interface to an underlying communication provider independent of the underlying communication provider. This interface is actually the set of library routines described below and developed for use by dnet datagram applications.

The first step that the DGS module set must perform is to register with the DGMS module. This is facilitated through the dn_cinit and dn_chandler library routines. These are the same library routines used by other handlers, although the dgms will check for a process registering with one the dgs module set reserved names and will interact differently in some situations. The dn_cinit call will register the dgs module set under it's reserved name, and will provide a means for passing datagrams to the dgs module set's representative network(s). The dn_chandler library routine is optionally used (currently only on dgsdec) to allow for asynchronous receipt of datagrams from the dgms.

After being properly registered, then the dgs module set is responsible for establishing a protocol dependent endpoint for communication with other dgs module sets of the same type on different machines. A name is bound to this endpoint for the peer dgs module sets to send to. At this stage a perceived full duplex connection exists through the dgs module set from its underlying network endpoint to its dnet endpoint.

The only responsibility left for the dgs module set is to maintain this full duplex connection, thereby providing the dgms access to the underlying protocol. To provide this, the dgs module set must respond to events on either side of the full duplex connection. For the UDP/IP and TCP/IP module sets there actually exists two modules, one for reading from the underlying protocol and passing datagrams on to the dgms, and another for reading datagrams from the dgms and passing them on to the underlying protocol. The DECnet module set is implemented with only one module, and uses the dn_chandler routine to respond asynchronously to datagrams coming from the dgms, while waiting for datagrams from the underlying protocol. The DECnet module was designed in this fashion so as to work efficiently on the VMS machine.

The specific details of implementation are discussed in the appropriate sections that describe every possible combination. This section will describe the general requirements that every DGS component must meet.

All dgs modules are invoked independently of the dgms, although they will fail if they are invoked before the dgms is running. The shell program: startdgms, and the DCL script: strdgms.com illustrate an acceptable method of network initiation. A more detailed discussion of the network initiation should be found in the administrator's guide.

4.3 DataGram Master Server (DGMS)

The dgms module provides two basic functions for the datagram services: routing and multiplexing of datagrams.

4.3.1 The Routing Function

The routing function is driven off of the same table used by the connection services for routing, although some different fields are used. The following is a description of this routing table:

destnet	nexthost	relay	nextproto

The destnet field is the primary key to this table. The nexthost field is the name of the gateway machine that the datagram should be sent to in propagating the datagram towards it's destination. The relay field is not used by the datagram service. The nextproto field describes the protocol of the network that is between the current host and nexthost. The values that this field may contain are currently "tcp" and "dec".

The fields within the user datagram structure are used to determine which record to pull (if any) from the routing table. The following structure describes the user datagram:

```
struct node
{
        char host[I_MAXHNAME];
        char net[I_MAXNNAME];
        char proc[I_MAXPNAME];
};

struct udg      /* User Datagram structure */
{
        struct node src;
        struct node next;
        struct node dest;
        long maxhops;
        int type;
        long buflen;
        char buf[1];
};
```

The user datagram structure provides a definitive description of a datagram. The user library stamps the src node information on the way out (the dgs module sets avoid this stamping with a special flag on the dn_cwrite routine). This information is not used directly by dnet components, but rather is there to provide the dnet application with information about the datagram source. The destination node is set by the user application and is never modified by the dnet components. The dgms module reads this information in it's routing function and sets the next node field accordingly. The next node field is used only to instruct the immediate dgs module set as to where to send the datagram to next. The next net field is currently never used by the dgs module or set by the dgms module. This requires that machines be named uniquely when there is a possibility that they will reside on networks common to any gateway machine.

In addition to the above, the dgms module must be able to correctly describe itself in terms of machine name and connected networks. This is determined from the myname table which is described below:

hostname	netname

The myname table is merely a list of all directly connected networks. This is used by the dgms to determine when a datagram has reached it's destination network. (The network protocol type must be determined by looking in the net table.) The hostname field is repetitive, but is required for the dgms to ascertain when a datagram has reached it's destination machine.

The process for routing is, then, as follows. A datagram arrives at the dgms, the destination node is checked against the myname table to place the datagram in one of the following three categories:

- The datagram is at the destination host and network
- The datagram is at the destination network but is not yet at the destination host
- The datagram is not yet at the destination network

Datagrams in the first category need no further routing and hence are passed on directly to the multiplexor. Datagrams in either the second or third category require a network table lookup, using the **dest.network** subfield of the datagram as the key. After a successful lookup, a table lookup is performed on the **next.proto** subfield of the network table to determine the name of the dgs module set process to send the datagram to. In addition, if the datagram is in the third category, then the **next.host** subfield of the network table is used to determine the name of the gateway to send the datagram on to. The **next.host** subfield and converted dgs process name are placed in the next node field of the user datagram, which is then passed along to the multiplexor.

4.3.2 The Multiplexor Function

The multiplexor performs a simple table lookup using the **next.proc** subfield of the user datagram against the Active DataGram User Table (ADGUT). Datagrams from the first routing category will have the name of the destination process in this subfield, whereas datagrams from the second and third routing category will have the process name of the dgs module set necessary to send the datagram on in its appropriate direction. The ADGUT is described below.

All users of the DNET datagram service must be registered with the DGMS and entered into the ADGUT. The DGMS will insure that two processes may not bind to the same process name. The process name to be bound to the datagram communications endpoint is specified in the call to dn_cinit. The dgms is informed of the process name and other vital administrative or control information through its service interface.

The service routines to the dgms are accessed via the same Inter Process Communication (IPC) mechanisms that are used to send and receive datagrams. This means the dgms has only to contend with one I/O descriptor for reading. This requires one more level of abstraction above the datagram. The unit of this abstraction is referred to as a dgms message. The dgms message is used to encapsulate either a datagram or a service request. The following structure describes the dgms message.

```
struct dgms_msg
{
        int type;
        long buflen;
        char buf[1];
};
```

The type field identifies the contents of the buf field and may be one of the following values:

D_MSGDG	User Datagram
D_MSGSRQ	Service Request
D_MSGSRS	Service Request Response
D_MSGSHD	Shutdown Advisory -- Not currently implemented

4.3.3 The DGMS Service Routines

The dgms service routines mentioned above provide a means for dnet datagram applications (including the dgs process sets) to interact administratively wit the dgms. The interface to the service routines are provided through the dgms_serv library routine.

The dgms_serv library routine issues a Service Request message to the dgms and awaits a Service Request Response message. The service request and service request response are both data structures. Your program is required to fill out the service request structure before calling dgms_serv. After returning from dgms_serv, the calling program interprets the results left in the service request response structure. The following describes these two structures:

```
struct srvreq
{
        int service;        /* service token */
        int pid;  /* process id of requesting process */
        char pname[D_MAXPNAME];/* requested process name to be bound */
        char rspipcname[D_MAXPATHNAME];/* set only by dgms_serv */
        char ipcname[D_MAXPATHNAME];/* where to send datagrams */
        int value;          /* service dependent field */
};

struct srvrsp
{
        int service;
        int pid;
        char ipcname[D_MAXFNAME];/* no longer used */
        char pname[D_MAXPNAME];/* not used */
        int retval;         /* return value */
};
```

The following is a list of available services. These services are described in detail later in this document.

DN_REQBAS	Request Basic DGMS Service. Register a process name, send datagrams, and receive datagrams synchronously.
DN_REQLIS	Request Listen DGMS Service. Receive datagrams asynchronously.
DN_REQCLN	Request DGMS Cleanup Service. Free up any resources tied up by the identified communications endpoint.
DN_REQAGS	Request ADGUT Status Service. Receive a copy of the ADGUT table in its entirety.

Although the service request message is sent through the same IPC mechanism used for sending datagram messages, the service request response messages are sent through a separate, transient IPC mechanism. This is due in part to the service routines being used to actually establish the IPC mechanism used for receipt of datagrams.

The dgms_serv routine attempts first to establish an IPC endpoint for receiving the service request response. The service request always uses the same ipcname (dgmsrs) and will make multiple attempts to gain access to this ipcname in the UNIX environments. In a VMS environment, this will only happen to processes under the same login session, because normal user processes do not advertise their logical mailbox name, but rather their actual mailbox name through the service request (the vms version of ipcget changes the logical name passed to the actual mailbox name) to this rule is the dgms, who uses a special flag (and must have SYSNAM privilege) on the ipcget routine to advertise the logical name in the system table.

After an appropriate IPC endpoint has been established, the dgms_serv routine assigns the ipcname bound to the endpoint (mailbox device name in VMS) to the **rspipcname** field of the **srvreq** structure. The service request is packaged in a message and sent out the standard IPC mechanism used for sending to the dgms. A blocking read is then performed on the newly created IPC endpoint waiting for the service response. After a service response is received, the endpoint is freed (possibly making it available to other processes) and the service response is returned to the function calling dgms_serv.

The following describes in detail each of the service routines supported by the dgms:

Request Basic DGMS Service

This request is made by the **dn_cinit(3U)** user library routine and performs three basic functions:

1. Establish an entry in the ADGUT table to describe the datagram communications endpoint.
2. Establish a means of sending datagrams to the user program by connecting to the IPC endpoint specified in the service request structure for receipt of datagrams. The user program must have already created this IPC endpoint before issuing this request.
3. Bind the process name specified in the service request to the newly created datagram communications endpoint.

The following fields of the **srvreq** structure are significant in the DN_REQBAS service request:

service = DN_REQBAS

pid This is the unique process identifier to be used when communicating back to this process information on the requested service (see below).

pname This represents an optional process name that the process wishes to be bound to so that datagrams sent will have a known process name. If no name is given, then the system will not send datagrams to this process.

value This represents the maximum number of bytes that this process is capable of receiving. If a message for this process is larger, it will be truncated, but the size field will not be altered so that the receiving process will know that there was information lost. THIS IS NOT CURRENTLY SUPPORTED.

ipcname This character string represents the name that may be used by the dgms to access the IPC endpoint so that datagrams may be sent to the user process. In UNIX environments this is a file name stored in a standard directory location. In VMS, this is the fully qualified pathname of the

mailbox being used for IPC. The requesting process should already have this IPC endpoint established.

rspipcname This is an ipcname in the same form as ipcname which is used to send the service response back to the requesting process. The requesting process should already have this IPC endpoint established.

Response Basic DGMS Service

The following fields from the srvresp structure are used:

service = DN_REQBAS

retval

The retval field may contain the following values:

0 Successful. The Request Basic DGMS Service control statement has completed succesfully and the DNET datagram user is now in a state where datagrams may be sent and received synchronously. The DGMS listen service may also be requested now.

-1 Internal DNET error. An internal error has occurred.

-2 No DGMS resources. There are currently no available entries in the ADGUT.

-3 ADGUT quota exceeded. You have exceeded the maximum number of entries you may use from the ADGUT table. This is not implemented yet due to the fact that only one endpoint per process may be established.

-4 No ipcname. The ipcname you specified for receipt of datagrams does not exist, or cannot be accessed by the dgms.

-5 Name in use. The process name that you requested to be bound to your endpoint is already in use by another process.

Request Listen DGMS Service

This control routine allows a signal number to be defined (in UNIX environments) to be used to inform the user application of a pending datagram. This routine has no real functionality for the VMS environment except to reset the state indicator for this process in the ADGUT.

service = DN_REQLIS

pid The primary use of the pid is to allow the DGMS to signal or interrupt the DNET Datagram User to indicate that a datagram has been received. This field should be the same as was specified in the DN_REQBAS request, as it will be used to query the ADGUT.

pname This field should be the same as was specified in the DN_REQBAS request, as it will be used to query the ADGUT.

ipcname This field should be the same as was specified in the DN_REQBAS request, as it will be used to query the ADGUT.

value

This is the "signal" (in UNIX terminology) that will be used to wake up the dn_handler routine. This field is required, but is not meaningful in s VMS environment.

Response Listen DGMS Service

This control statement will be initiated by the DGMS after receiving a Request Basic DGMS Service control statement. The Response Listen DGMS message will use the following fields of the srvresp structure:

service = DN_REQLIS

retval

The retval field may contain one of the following values:

0 Successful. The Request Listen DGMS message was serviced successfully and the calling process is now in a state associated with the Listen DGMS Service.

-1 Internal DNET error.

-2 Bad argument(s). The specified pid field was less than zero, the pname field was not specified, or the ipcname field was not specified. The dgms cannot perform the ADGUT query without these fields.

-3 An ADGUT entry was not found with the values supplied for pid, pname, and ipcname.

Request DGMS Cleanup The Request DGMS Cleanup message instructs the DGMS to free up all resources allocated for the process using the given pname, and under the provided pid. This will remove any unique IPC mechanisms associated with this process, if the IPC mechanism is not being used for another process name within the given process.

The Request DGMS Cleanup message uses the following fields from the srvreq structure:

service = DN_REQCLN

pid This is the actual process identifier. It will be used to determine which entries to remove from the DGMS Active Datagram User Table when a process name is being shared by more than one process.

ipcname This should be the same as used in the DN_REQBAS request, as it will be used for an ADGUT query.

Response DGMS Cleanup The Response DGMS Cleanup uses the following fields of the srvresp structure:

service = DN_REQCLN

retval The retval field will indicate the success of the Request DGMS Cleanup statement. A value of 0 will indicate success, and indicates that all resources being tied up by this pid, pname combination are now freed. The following values will indicate the error condition existing:

-1 Internal dnet error.

-2 Bad argument. The pid field or the ipcname field were not specified or were invalid.

-3 An ADGUT entry could not be found with the specified pid and ipcname.

Request ADGUT Status The DN_REQAGS service routine places a copy of the entire ADGUT into a datafile in the dnet_home directory. The service response data structure contains the name of the file that the ADGUT was copied into. The following fields from the

srvreq structure are significant:

service = DN_REQAGS

pid Process identifier.

rspipcname For sending service response.

Response ADGUT Status The response to DN_REQAGS includes the following fields of significance:

service = DN_REQAGS

ipcname The name of the datafile in the dnet_home directory which contains the ADGUT copy. The table is in its binary form and can be accessed using the **dgms_adut** structure defined in dgms.h and described below.

4.3.4 The ADGUT

The DGMS Active Datagram User Table is created and maintained internally by the DGMS so as to keep track of all processes that interact with it (including all DGS components).

```
struct dgms_adut
{
        int     pid;    /* Process Identifier */
        char    pname[MAXPNAME];/* process name bound to */
        char    ipcname[MAXFNAME];/* IPC name to use to send */
        int ipcid;      /* IPC id used to send messages */
        int maxmsg;     /* maximum size of message this component can receive */
        int signal;     /* Signal number used to inform of pending datagrams */
        unsigned w_timeout;/* timeout period on write */
        time_t  add_time;/* time entry was added */
        time_t  last_access;
        time_t  last_update;
        time_t  last_send;
        time_t  last_recv;
        int     state;  /* 0 - invalid, 1 - basic, 2 - listen */
};
```

Figure 1. DGMS Active Datagram User Table

The pid field is the process id of the process used primarily so that signals or interrupts may be sent to the process to inform it of impending datagrams. The pname field is a process name that is bound to the process. This allows outgoing datagrams to have a process name, and allows for incoming datagrams to be routed to the proper server process. The DGMS will allow only one process to be listening on a given process name, although many processes may be sending under a common name.

The ipcname field is used to keep track of the name of the IPC mechanism used to send messages to that particular process. The ipcid is used to hold the id (a file descriptor in the case of named pipes) of the IPC mechanism.

The maxmsg field is not currently supported but is intended to allow a user application to impose limits on message sizes that may be passed to them. This is handled now by requiring that all user processes

be capable of handling the maximum size message, or the biggest message they expect to receive.

The signal field is used only in UNIX and indicates the signal number that is to be sent to a user application in the listen state when a datagram is pending.

The w_timeout field is always set to 0 except for special dnet processes who will have a hard-coded value. This value represents the amount of time (units are system dependent) that the dgms will block on a write attempting to send a message. In the DGS module sets, this time is intended to represent the normal amount of time required to relieve itself of a datagram. On all other users applications, the datagram will be discarded if the message cannot be sent immediately. A terse error message will appear in the dgms log file to indicate that this occurred.

The state field indicates the current state that an endpoint is in. This field is used when determining if an ADGUT entry is available.

The time fields are all used to monitor activity of the user applications.

4.4 The Connectionless Services Library

The connectionless services library consists of seven user function calls:

dn_cinit	Establish endpoint and basic service state
dn_cwrite	Send a datagram towards a destination node
dn_chandler	Declare an exception handler for asynchronous receipt of datagrams
dn_cread	Read a datagram synchronously
dn_cdone	Free up datagram communications endpoint resources
dn_salloc	Dynamically allocate dnet data structures
dn_cerror	Send a dnet error message (including stack trace) to stderr

4.4.1 The Function Of dn_cinit

The dn_cinit library routine places the DNET Datagram User into a state associated with the Basic DGMS Service. This involves establishing a dnet communications endpoint and binding a process name (possibly null) to that endpoint with the DN_REQBAS service routine. The dn_cinit routine also establishes the IPC mechanisms necessary for sending and receiving of datagrams.

A user application that is in a basic state is capable of sending a datagram to another registered user application (local or remote). In addition, the user application in a basic state may request the listen state as long as a valid (non-null) process name was bound on the datagram communications endpoint.

4.4.2 The Function of dn_cwrite

The dn_cwrite function call facilitates the sending of a datagram to a remote process. This is done by source stamping the datagram (filling in the source node of the datagram structure), encapsulating that datagram in a dgms message structure and passing that message along to the dgms. No reliability is implicit or explicate within dn_cwrite.

4.4.3 The Function of dn_chandler

The dn_chandler library routine is by far the most complicated and operating system dependent of the function calls. The basic function of the call is to place the user into a state associated with Listen DGMS Service. This involves identifying an exception routine that is to be called when a datagram has arrived for this user.

The UNIX implementation will allow the address of the exception routine to be identified to the library routines. The library routines will set up a trap for the signal that the user specifies. The signal can not be used for any other purpose within the user application. This is why the dn_chandler

function call allows the process to choose the signal. This information will be passed along to the DGMS so that it may be included in it's Active Datagram User Table.

After the routine and signal have been set, then the receipt of the datagram for this process will result in a signal being sent to the process by DGMS (this of course requires that DGMS have an effective uid of root), which will then cause the library routine to call the exception routine specified in the original call. The exception routine will be passed the address of the **udg** structure containing the datagram just read.

The VMS implementation will be similar, in that an exception routine is specified, and the address of the **udg** structure is passed as an argument to the exception handler when a datagram arrives. The VMS implementation will, though, use asynchronous traps (AST) to inform of pending datagrams.

4.4.4 The Function of dn_cread

The dn_cread library routine provides for the synchronous receipt of datagrams. The dn_cread routine is actually called as part of the handler function of dn_chandler. This routine reads a datagram that is pending. The routine is capable of working in either blocking or nonblocking mode (the default is blocking).

The dn_cread routine basically maps directly to a ipcrcv call, checks the message type for a datagram, and if so, unpacks the datagram and passes it back to the calling function.

If the message type read was D_MSGSHD, then a dn_cdone is issued, the dnet_errno value is set to D_SHUTDOWN, and the call returns in error (with a -1).

4.4.5 The Function of dn_cdone

The dn_cdone routine provides for the freeing of resources normally allocated for a datagram communications endpoint. This is done mostly through the DN_REQCLN service routine.

The dn_cdone routine itself also closes any standard IPC mechanisms, and removes the mechanisms that it created (inherently by the ipcclose). Finally, the dn_cdone routine resets any signal handlers activated by a call to dn_chandler.

4.4.6 The Function of dn_salloc

The dn_salloc library routine is used to dynamically allocate dnet data structures that contain an imbedded buffer to be used for the layering of abstractions. This function is useful so that a program may choose the size of the buffer rather than always creating a buffer of maximum size, and can also be used to create reentrant/recursive code sections (possibly in combination with the dn_chandler routine).

The dn_salloc uses the malloc routine to allocate new memory. The dn_salloc is used in many places throughout the code, but is never used in attempt to create reentrancy of the services code, but rather to create a standard mechanism for obtaining raw space for data structures.

4.4.7 The Function of dn_cerror

The dn_cerror function is used merely as a last resort to try to display diagnostic information as to why something failed. A stack trace of dnet calls is displayed to try to provide insight as to where in the user code the failure occurred.

The stack trace is facilitated within the datagram service code through an array of character arrays that hold the name of the library routine called. This array is updated by calling macros defined in the dnet_errno.h file. These macros are:

- DE_push()
- DE_pop()
- DE_print()

4.5 Component Interaction Diagrams

The following diagram describes schematically all of the components of the datagram service that have been discussed. Following that diagram, there is a series of diagrams describing the series of steps that are taken in bringing the network up and in sending a receiving a datagram.

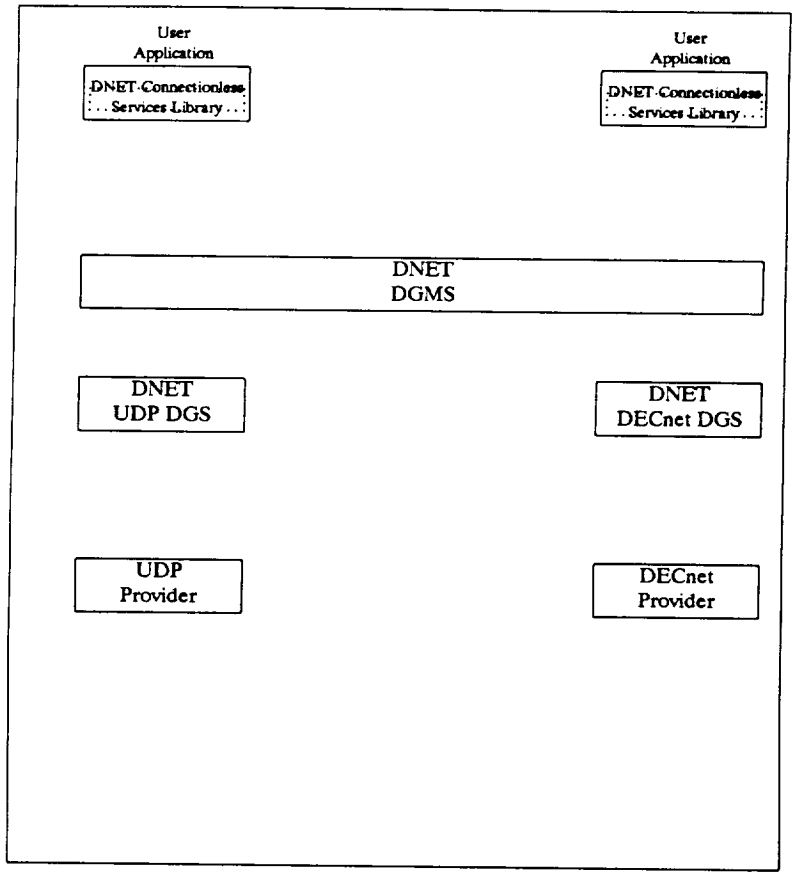


Figure 2. Schematic Overview of Connectionless Service

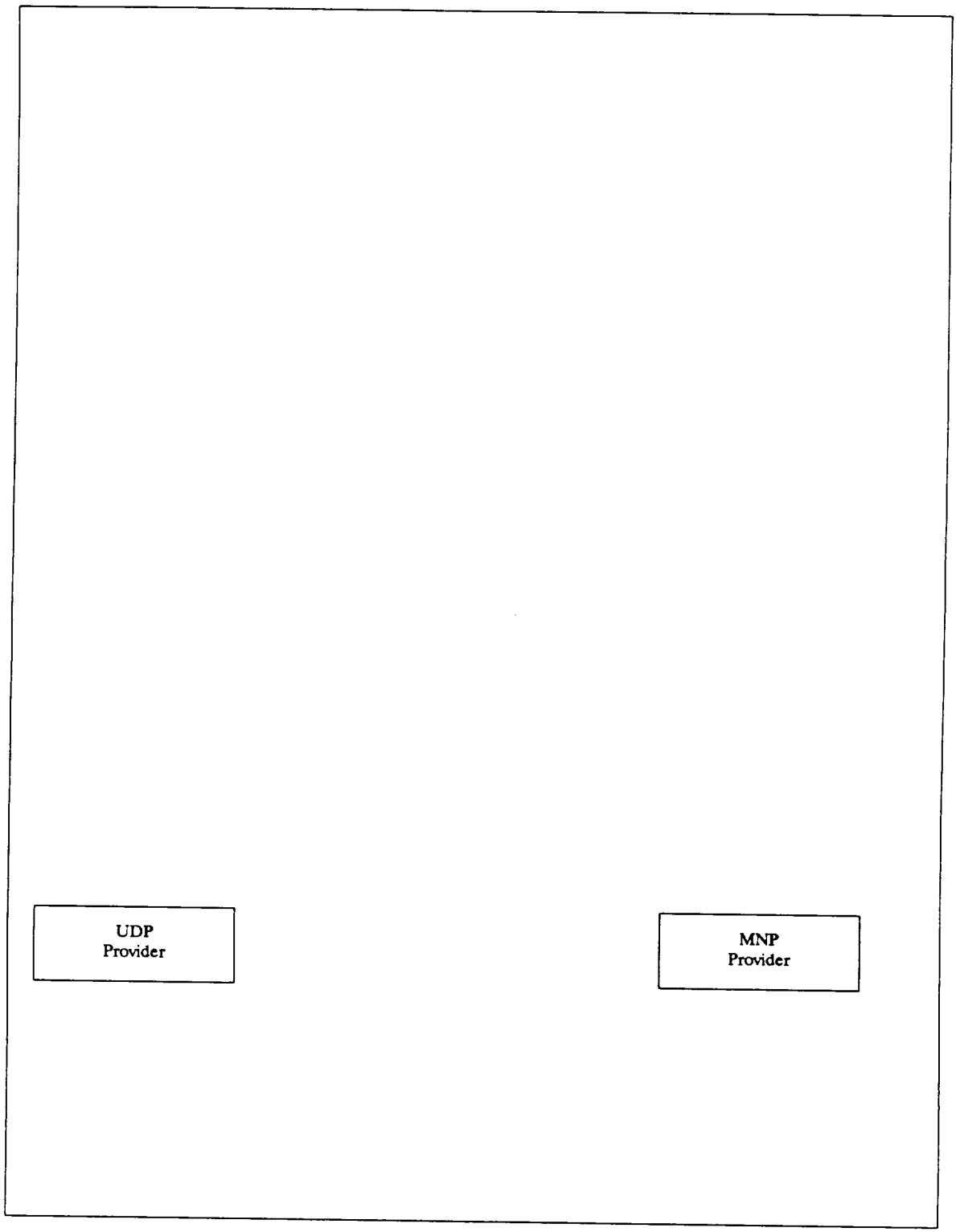


Figure 3. Empty Host Machine With No DNET Components

This represents an empty machine. The only components of interest existing on the machine are any underlying protocol providers.

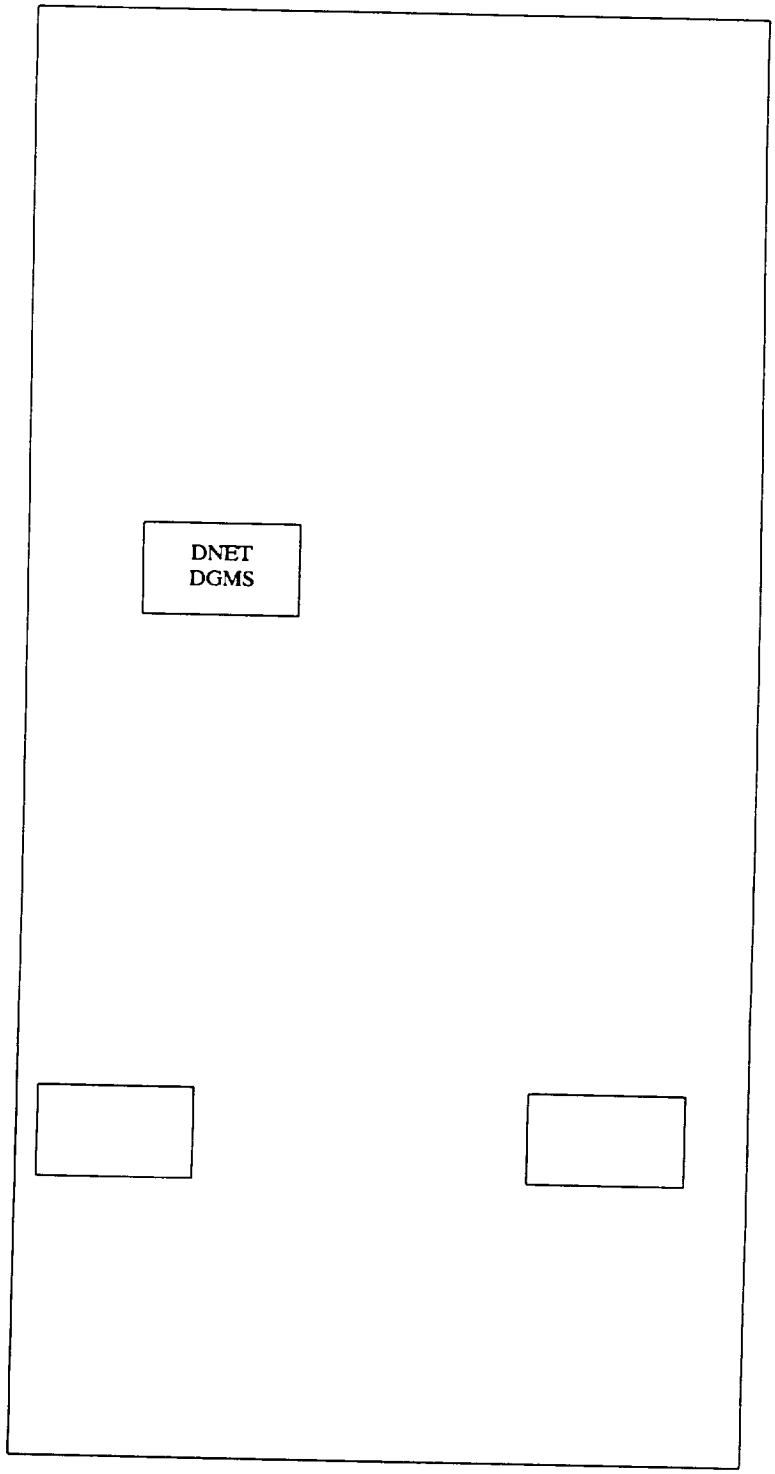


Figure 4. Datagram Master Server Started

Here, the DataGram Master Server is started either manually by a systems administrator, or through a regular boot up procedure in the machine. The DGMS will coordinate all connectionless service activity and so will be the first component started.

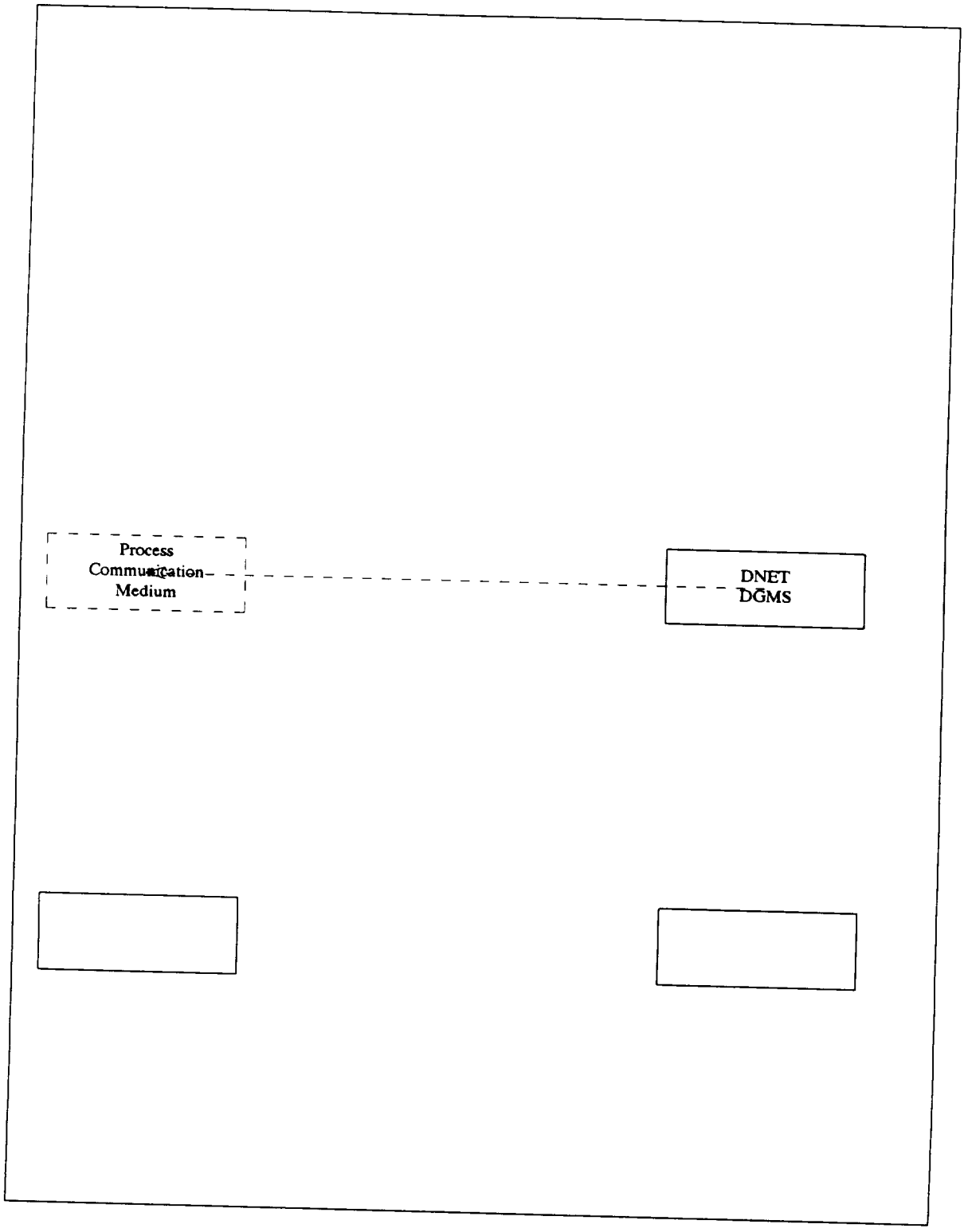


Figure 5. Process Communication Medium Preparation

The DGMS will first create the process communication medium. This will be one form of communication that will be used by all components when interacting with the DGMS. This will allow the DGMS to concentrate on reading from only one entity. The communication medium must support message oriented service. The message oriented service will provide for the synchronization of otherwise potentially non-atomic writes over a single IPC mechanism potentially shared by many writers.

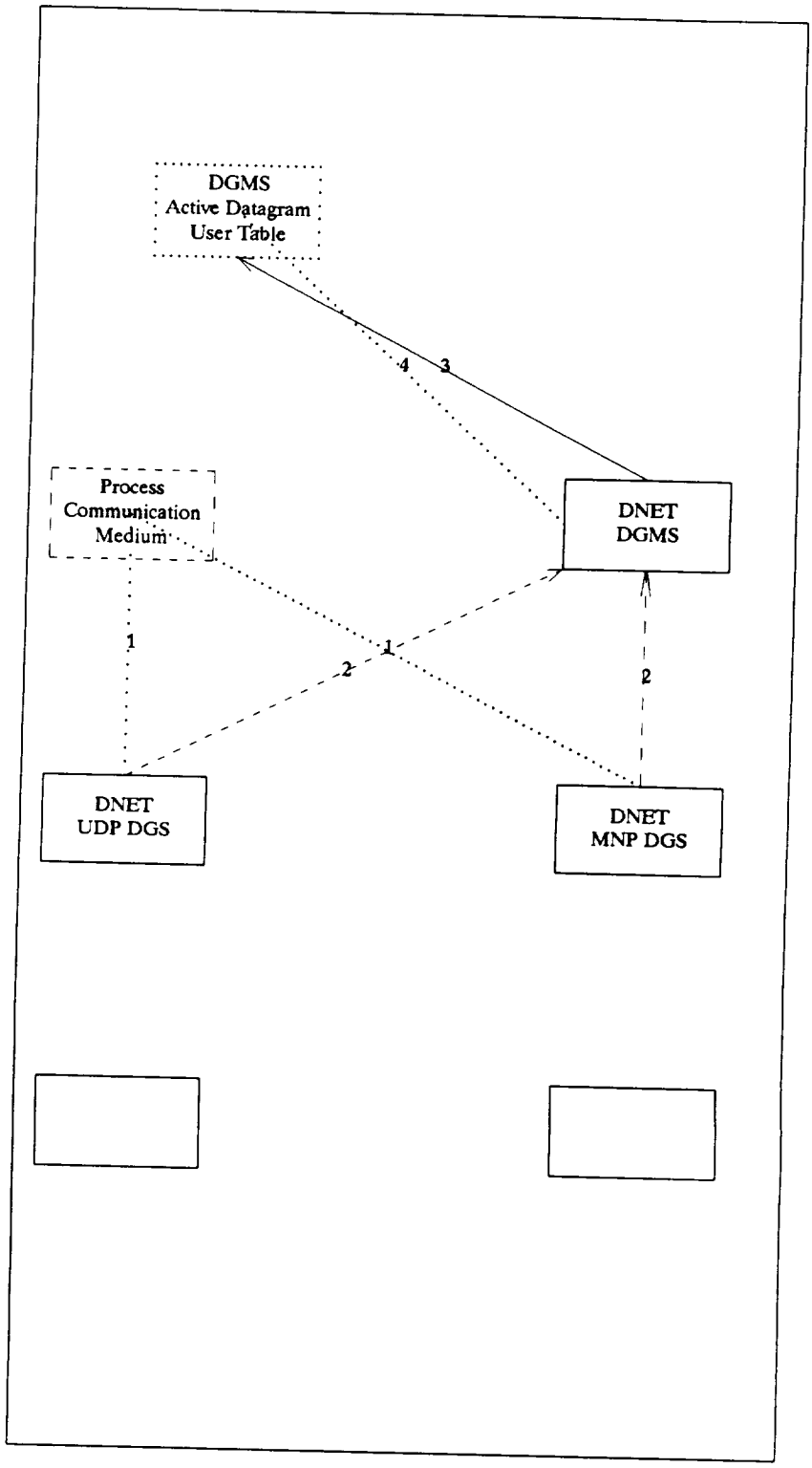


Figure 6. Invocation of DGS Components

1. The DGS program sets are started independently of the dgms program. These issue the dn_cinit call, which establishes the IPC communication endpoint for receiving datagrams and connects to the standard IPC communication endpoint for sending messages to the dgms.
2. After the connection to the dgms is made, service request (DN_REQBAS) is made (via dgms_serv).
3. The dgms responds to the service request by establishing a dnet datagram communications endpoint in the ADGUT and binds the requested process name (**dgstcp** and **dgsdec** in this case) to the established endpoint.
4. Finally the dgms connects to the IPC endpoint created by the dn_cinit routine in step 1. This simplex connection will be used to send datagrams to the dgs program sets. The service response is sent through a transient IPC mechanism created and maintained by the dgms_serv routine.

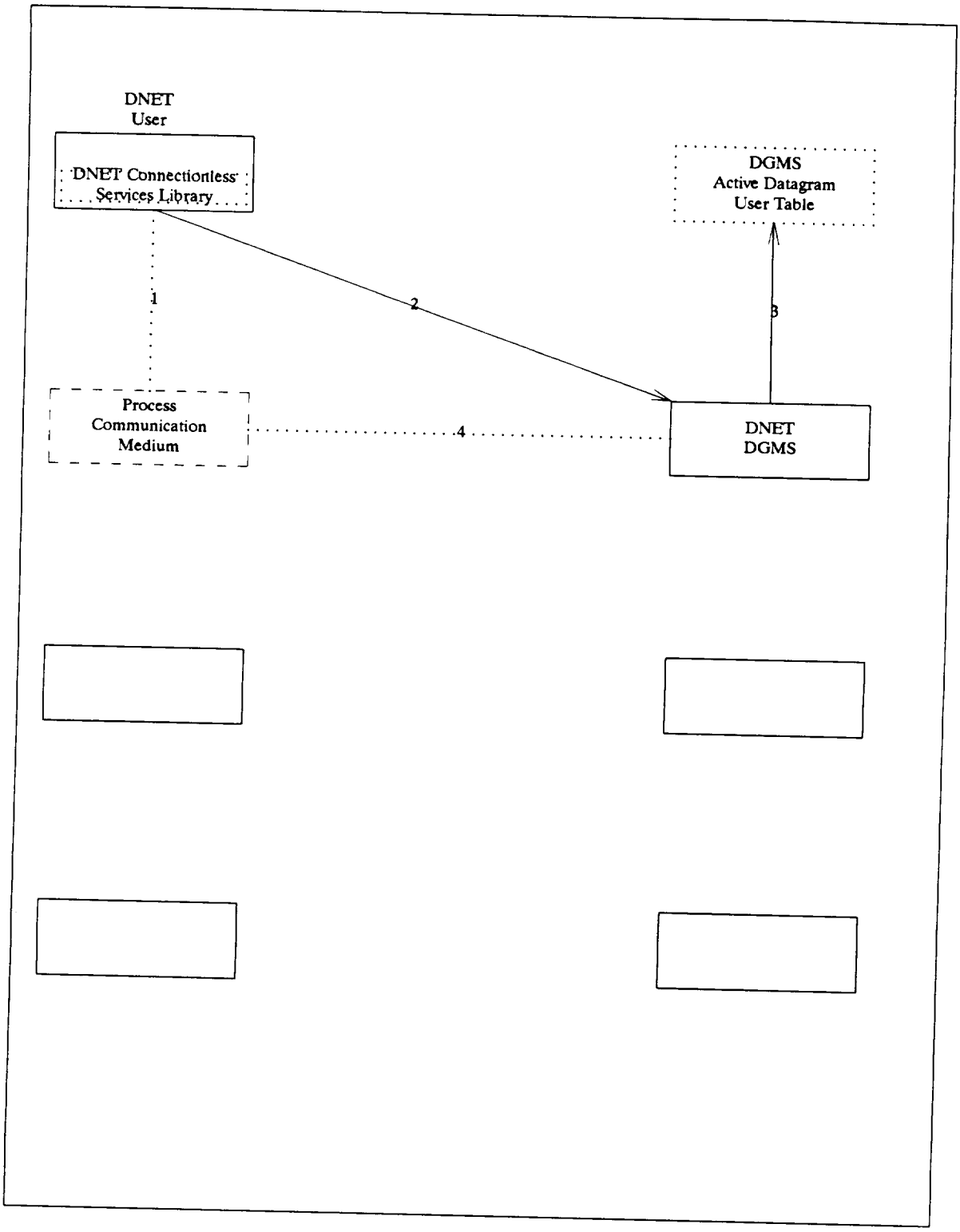


Figure 7. dn_cinit process

The dnet user application uses the same procedure as the DGS program sets in accessing the dgms.

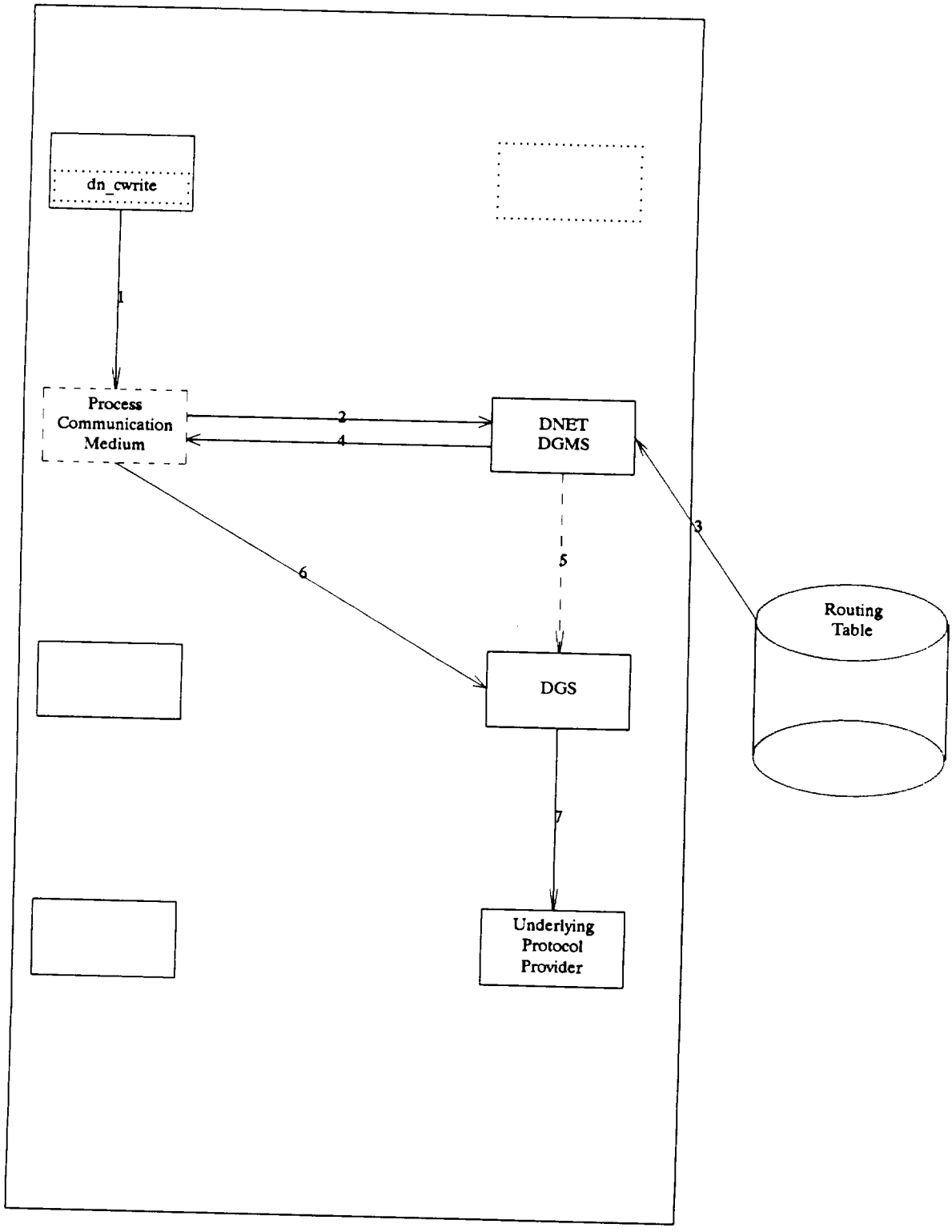


Figure 8. dn_cwrite Process

1. The dn_cwrite will send a message containing a datagram with a DGMS Message Header inserted (exactly the same way that the DGS component passes datagrams to the DGMS and through the same key value) onto the process communication medium.
2. The DGMS (again who is performing a blocking read on the process communication medium) will read this message and will interpret it as being a datagram.
3. The DGMS component consults the routing table to determine the address of the next hop (after determining that the destination is not the current machine... again the exact same mechanism used when a datagram arrives from a DGS component).
4. The DGMS sends a message out to the process communication medium of type datagram and sent under the key value so that the proper DGS component will read it (this is determined from the routing table).
5. If necessary (in dgsdec only) a signal is sent to inform the module that a datagram is pending (the dgsdec module is blocking in the underlying protocol side). In the case of the dgsudp program set, one process is blocking on the UDP side, while the other process is blocking on the dgms side. No signal is sent in the latter case. The dgms is aware of this because the dgsdec module will have a state of 2 because of using dn_chandler, whereas the dgsudp program sets will have a state of 1 since they only used dn_cinit.
6. The DGS component reads the message from the process communication medium and prepares internal structures (structures that are unique to the appropriate protocol).
7. The datagram is then sent to the underlying protocol. All special considerations of underlying protocol are handled here. For example, if the underlying protocol does not support a connectionless service, then a connection is established for each datagram to be sent.

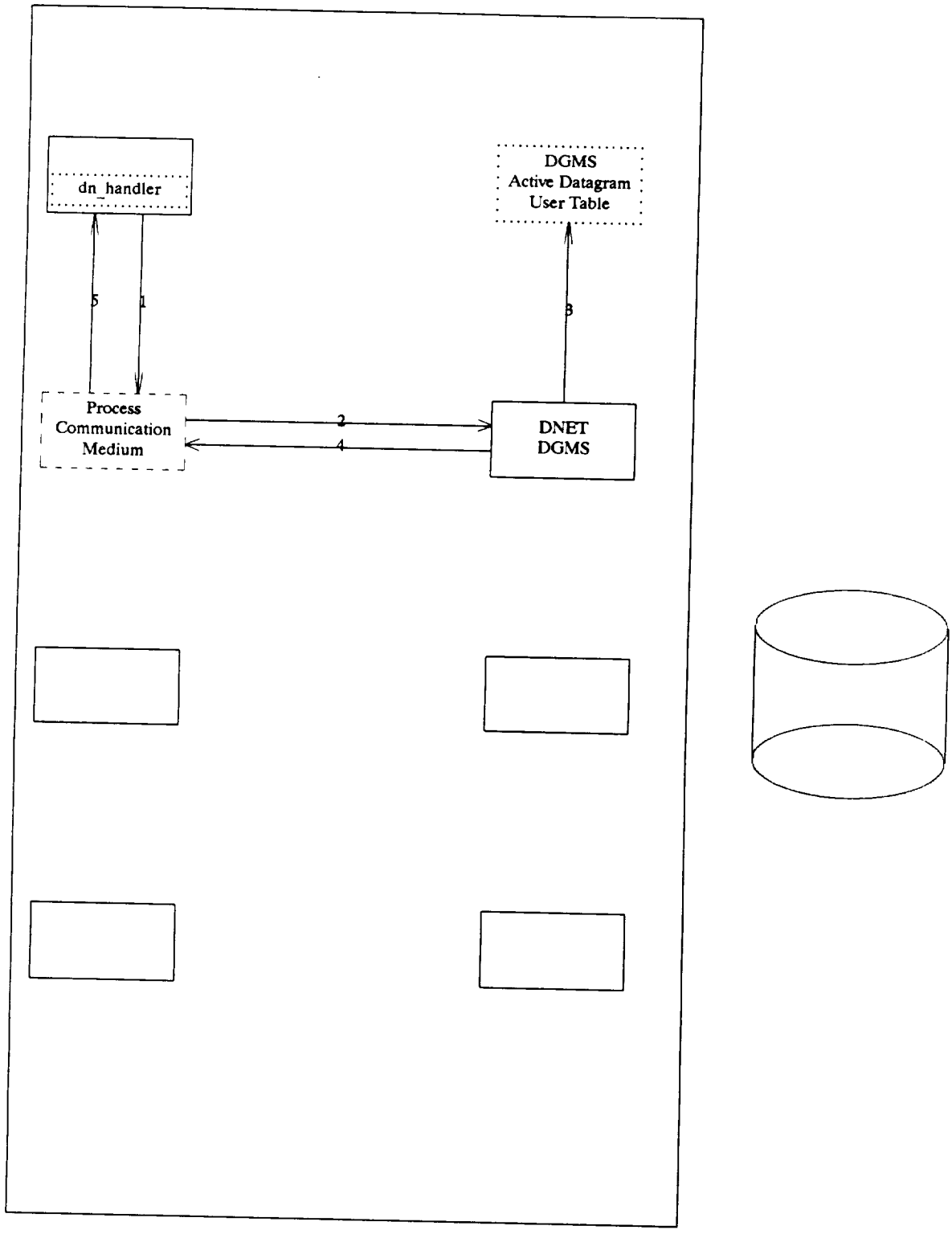


Figure 9. dn_chandler Process

1. After achieving the state associated with Basic DGMS Service, the DNET User component is able to move to the state associated with Listen DGMS Service. The dn_chandler sends a Request Listen DGMS Service message to the process communication medium under the hard coded key value used to communicate with the DGMS (the same value used when sending the Request Basic DGMS Service message).

2. The DGMS reads the message sent to it and interprets it as being a Request Listen DGMS Service message.

3. Assuming the component sending the Request Listen DGMS Service message is in the proper state (it must have previously sent a Request Basic DGMS Service message and be listed in the DGMS's Active Datagram User Table), the DGMS will modify the entry in the Active Datagram User Table.

4. The DGMS sends a Response Listen DGMS Service message out to the DNET Datagram User with the ipcname specified in the Active Datagram User Table. Information included in this message includes the DNET User Identifier, and key value which may contain a negative number indicating an error.

5. The dn_chandler routine will have been waiting for the Response Listen DGMS Service message over the standard response ipcname (this all happens in the dgms_serv internal library routine). After reading a successful response, the DNET Datagram User will now be in a state associated with the Listen DGMS Service and is capable of sending datagrams as well as responding to datagrams sent to it.

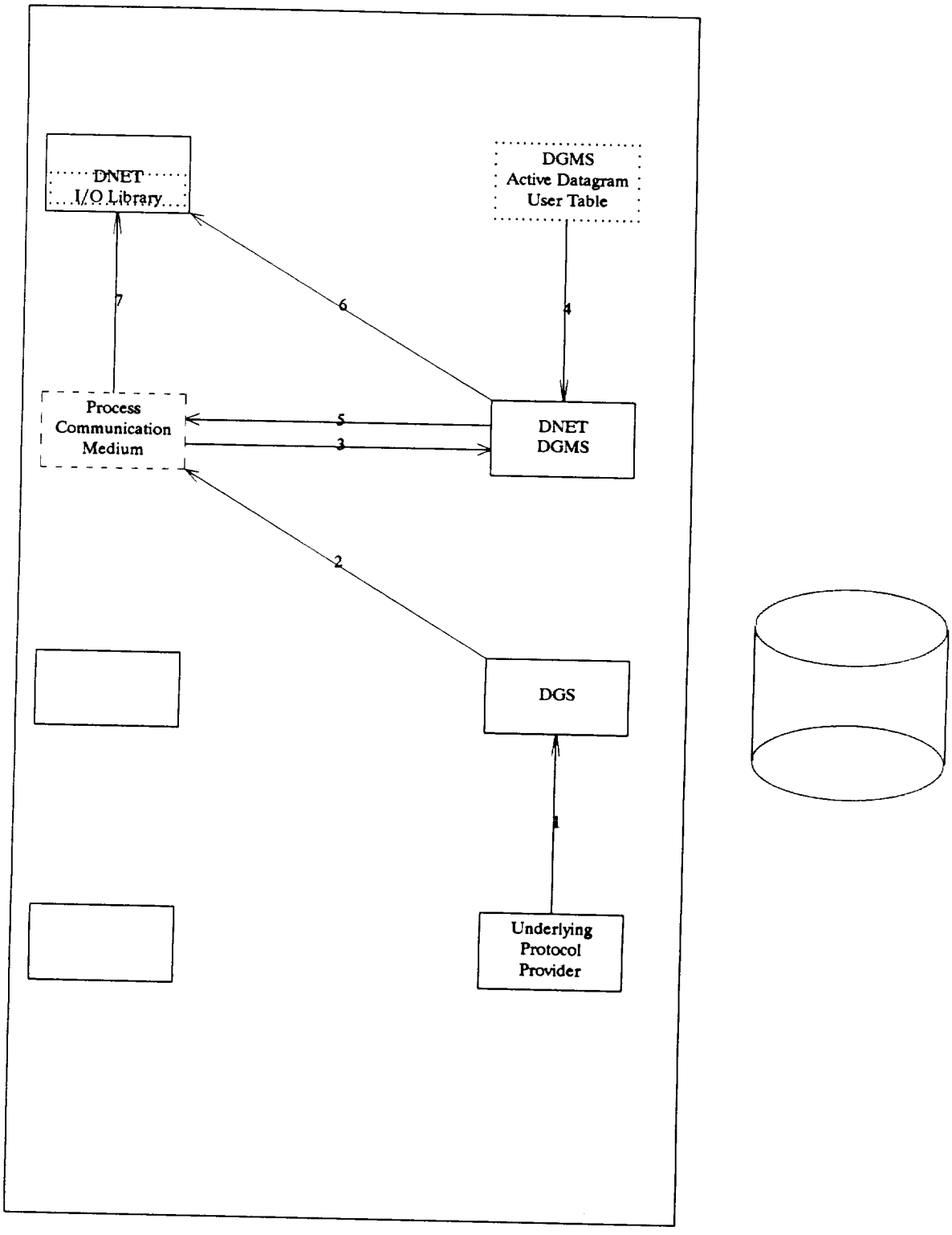


Figure 10. Receive Datagram at Destination

1. When a datagram arrives from another host, the underlying protocol passes it to the DGS component performing the blocking read on that particular protocol.
2. The DGS component inserts the DGMS Message Header on the datagram to form the datagram message (the exact message type that the DNET User component forms when sending a datagram with dn_cwrite) and this message is placed on the process communication medium with the key value specified so that the DGMS will read the message (the same key value used by the dn_cwrite routine when sending it's datagram message).
3. The DGMS reads the message from the process communication medium and interprets that it is a datagram message.
4. The DGMS figures out that that the datagram is destined for this host, and so it checks the DGMS's Active Datagram User Table to find the process (if one exists) that is waiting to receive this datagram. The DNET process ID and the key value for this process will be pulled from the Active Datagram User Table.
5. The message is placed on the process communication medium with the key value specified so the proper DNET User component will read it.
6. If the state associated with the intended destination of the datagram is 2, indicating that it received datagrams asynchronously, a signal is sent when the operating system environment is UNIX.
7. The DNET User component will read the message waiting for it on the process communication medium.

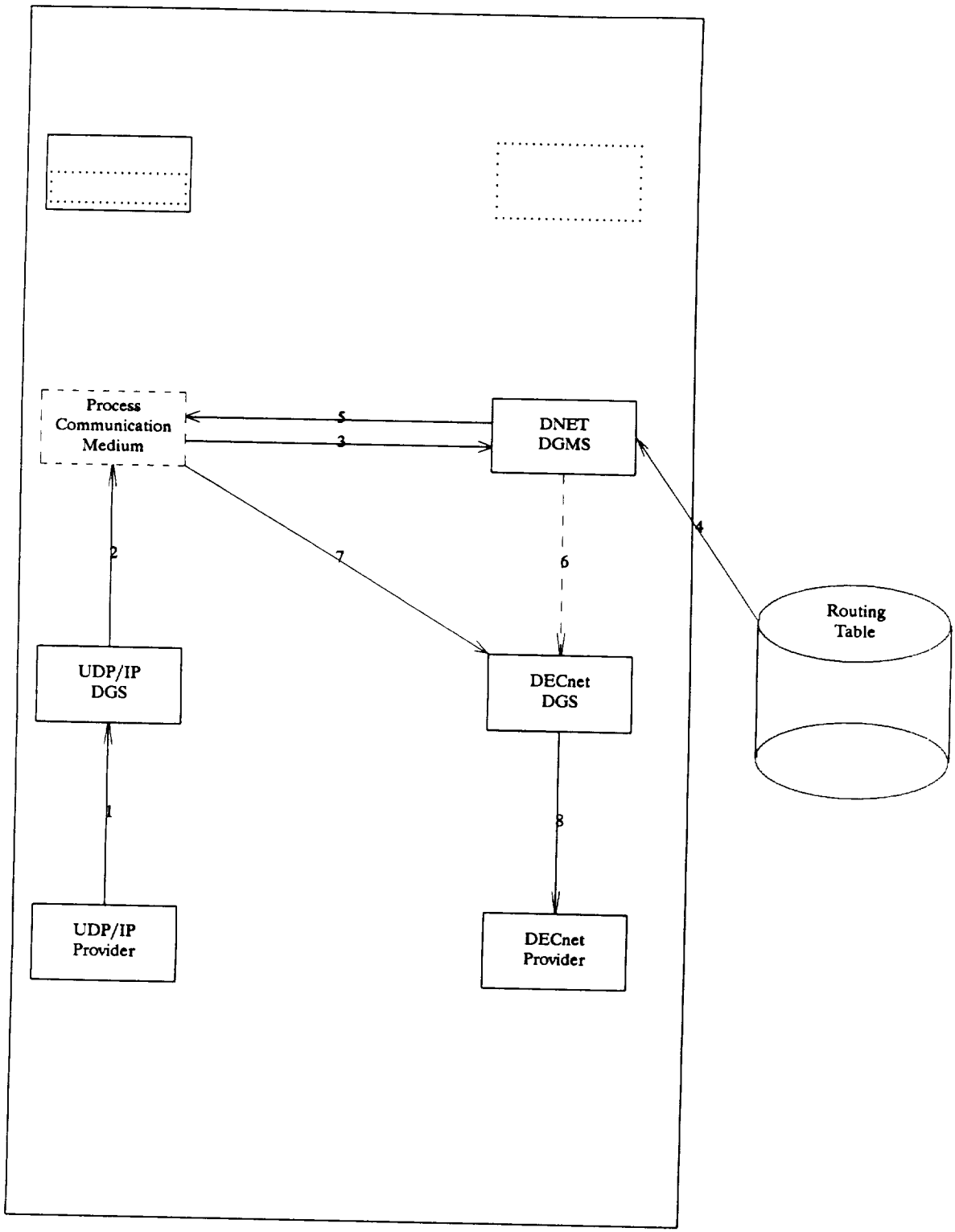


Figure 11. Receive Datagram: at Gateway

1. A datagram is received by the underlying protocol and is passed along to the DGS Component associated with that underlying protocol.
2. The DGS component inserts the DGMS Message Header and places the message in the process communication medium with a key value such that the DGMS will read it.
3. The DGMS reads the message and determines that it is a datagram message.
4. The DGMS determines that it does not represent the destination machine, and so consults the Routing Table to determine the next hop.
5. The DGMS changes the next hop node in the user datagram structure to state the node information of the next hop and places the datagram message back into the process communication medium with a key value such that it will be read by the proper DGS component.
6. If necessary, the dgs program (dgsdec only) will be sent a signal if it resides on a UNIX machine.
7. The DGS component will then read the datagram message and prepare the internal structures in preparation for passing it along to the underlying protocol.
8. The DGS component will then pass the datagram along to the underlying protocol provider. As stated previously, any provision for support of connectionless service in an underlying protocol which otherwise does not support a connectionless service is the responsibility of that DGS component.

5. DNET Interprocess Communication (IPC)

5.1 Introduction

The standard IPC implementation was created to provide a standard means of communicating between processes running in varied environments. These means must be capable of providing services necessary and reasonable for both the connectionless and connection services. The following summarize the requirements placed upon the IPC services:

Independence from DNET

The mechanisms should serve all of the needs of the dnet services but should avoid (where possible) imposing dnet constraints. These constraints could be usage of global constants defined in the dnet domain, or reliance upon dnet locations within a file system, or the usage of functions defined within the dnet domain. THIS REQUIREMENT HAS NOT BEEN COMPLETELY MET. MOST VIOLATIONS OCCUR WITHIN THE VMS ENVIRONMENT.

Implemented under BSD UNIX, System V UNIX, and VMS

The standard IPC implementation should be accessed the same regardless of the operating environment under which it was created. This was the primary reason for creating the standard IPC implementation, so as to provide a standard interface for communicating with other processes on the same box.

Simplex IPC mechanisms

The IPC mechanisms established need only be simplex. This requirement is stated to allow for economizing resources. If a full duplex connection is required (the exception rather than the rule in dnet), then two mechanisms may be established. In two of the three operating environments, this does not require any more resources than an actual full duplex mechanism.

Message oriented transmission

The message oriented transmission is required mainly because of the need to have a single reader responding to many writers. The IPC mechanisms themselves are more capable of managing messages than is the receiver capable of making messages from a stream. All operating environments provide a direct IPC mechanism for passing messages.

Oriented towards endpoint establishment

The endpoint establishment should be contrasted with a "mid point" establishment such as the message queues used in System V UNIX. The mid point establishment allows a common area to be logically set up where messages may be placed (a bulletin board of sorts), all members are then able to pick any message sitting in the mid point. The endpoint method allows one process to advertise an address which can be globally sent to, but only locally read. The BSD socket interface is an example of endpoint establishment. The VMS mailbox devices would fall into the category of mid point. The endpoint

establishment requirement was set down mostly because it is easier to force a midpoint type IPC mechanism to act like an endpoint type IPC mechanism than vice-versa.

Addressing via character strings

All IPC endpoints should be addressable with character strings. There are some limitations to this including length and use of special characters like "/" in UNIX and ":" in VMS. The BSD and VMS provide direct addressing of endpoints with character strings. The conversion used for System V message queues is discussed in the section on implementation for System V.

Independence between peers

All interactions across the IPC mechanisms should be performed independently of the action or availability of the peer. If the buffer between the peers is full, then the sender should have the ability to fail the write without wasting time blocking. In no case should the success of the write be hinged upon the availability of a read.

5.2 Interface

The interface to the IPC mechanisms is intended to be similar to a socket interface, but simpler because of less required functionality. In addition, and more important, the interface should be independent of the operating system, although there still exist some subtle differences. Finally, there is a provision for binding and connecting (like the socket interface), but the simplex nature of the connections allows this provision to be included in the library routine to establish the endpoint.

The following, then, are brief descriptions of the library routines making up the general IPC interface.

5.2.1 Administration Of IPC Medium

To provide for full flexibility, it is necessary on some operating systems to create a medium for the IPC mechanisms. This medium may include an environment for name translation or location (UNIX) or may require that a chunk of the operating systems IPC "medium" be reserved for use by the set of cooperating processes (System V). The VMS operating does not require the administrative creation of the IPC medium, and in that environment merely no action is performed on an attempt to create the medium.

The phrase "midpoint establishment" merely provides convenient semantics for this discussion and should not be confused with any other phraseology that may be similar

Two routines are provided for the administration of the IPC medium. One routine is called to create the medium, and the other routine is called to remove or free the medium. An explanation of these two routines follow.

5.2.1.1 The _makeipc function

The _makeipc function allows for the administrative creation of an IPC medium necessary for establishing and using IPC mechanisms for communication. The following is a listing of the declaration of _makeipc:

```
int _makeipc(sv_msg_key, ipcdir, flags)
int sv_msg_key;       /* System V message key value */
char *ipcdir;         /* UNIX directory where addresses will reside */
int flags;  /* D_CREAT, D_EXCL */
```

The **sv_msg_key** argument is only pertinent in the system V environment (although a value may be passed in any environment without harm). This value is used to determine the lookup key value for the message queue that will be used. One message queue is shared for all IPC mechanisms in System V. Consult with your administrator or use the UNIX **ipcs -q** utility to determine available message queue key values.

The **ipcdir** argument is pertinent in both UNIX environments. This is an absolute pathname for the IPC directory. Both System V and BSD environments use a UNIX pathname for the fully qualified address of an IPC endpoint. All endpoint establishments and references will be made with a filename only (no "/"'s are allowed) that is relative to the **ipcdir**. Once the IPC medium has been established and is being used, an examination of this directory will reveal the active connections.

The flags argument is used to allow for the creation of the IPC medium. When this routine is called administratively, the D_CREAT flag should be set at a minimum to insure that you will create the environment if it does not exist. It is suggested that you use the D_CREAT in combination with the D_EXCL to insure that two administrative processes are not reigning over a single IPC medium.

The use of this call with neither of the flags set is valid and is used to merely access an existing IPC medium. This call is already performed transparently from within the dn_cinit routine discussed below.

The _makeipc routine called explicately for the creation of the IPC medium must agree in its arguments with the implicate call made by the later calls to ipcget. The implicit call is made within the ipcget using the makeipc (no preceding underscore) routine which in turn calls _makeipc as follows:

```
if(_makeipc(DNET_IPCKEY, DNET_IPCDIR, D_CREAT) == -1)
.
.
.
```

Notice how preprocessor constants are used to facilitate the agreeance on arguments. For non-dnet use, it is suggested that the dnet constants be defined (in the dnet_ipc.h file) for your own use and that the D_CREAT flag should be set to 0. This will allow you to call _makeipc explicately using the preprocessor constants as arguments along with the D_CREAT | D_EXCL flags, and then the ipcget routine will use the same values when accessing the medium created by the administrative process.

The following is an example code section on suggested method for the adminstrative creation of the IPC medium:

```
#include "dnet_ipc.h"
#define DNET_IPCKEY 1504
#define DNET_IPCDIR "/tmp/myapp"
.
.
.
if(_makeipc(DNET_IPCKEY, DNET_IPCDIR, D_CREAT | D_EXCL) == -1)
{
        fprintf(stderr, "_makeipc failed: dnet_errno(%d) errno(%d).0, dnet_errno, errno);
        return(-1);
}
.
.
.
```

5.2.1.2 The _removeipc function

The _removeipc routine is provided to clean up the IPC medium created with the _makeipc routine. No arguments are passed to the _removeipc routine because the IPC module keeps track of the arguments used to call _makeipc. The environment will only be removed from the system if the _makeipc routine was called by someone with both the D_CREAT and D_EXCL flags set. Only the process that actually created the segment will be allowed to remove it. All other processes will return immediately without error. The rationale for this is that since there is no way of determining which process actually created the medium (multiple processes may assume this), then it is not reasonable to assume which process may remove it.

5.2.2 Administration Of Individual IPC Mechanisms

The individual IPC mechanism is a logical device which provides for simplex transmissions betweening peer processes on a common machine. These mechanisms should be assumed to be reliant upon an existing IPC medium. The proper IPC medium will be accessed using the preprocessor constants described in the section above.

The responsibilities of the program in administering individual IPC mechanisms is the establishment of the endpoint and the cleaning up of the endpoint when the program is through with it.

Two routines are provided for these purposes: ipcget and ipcclose. A description of these routines follow:

5.2.2.1 The ipcget function

The ipcget function provides for the establishment of an IPC endpoint and provides for either a address to be bound to that endpoint, or a connection to be made to another endpoint with an address bound to it. After a successful ipcget, the endpoint is an established IPC mechanism and may be used for either receiving datagrams (if bound) or sending datagrams (if connected). No support for a connectionless endpoint exists where the address is specified on each message, and the only way to bind or connect to a different address is the remove the endpoint and reestablish.

The following is the declaration of the ipcget function:

```
int ipcget(name, flags)
struct dnet_ipcname *name;
int flags;
```

The dnet_ipcname structure consists of the following fields:

```
struct dnet_ipcname
{
        char name[D_MAXPATHNAME];
        unsigned maxmsg;
        unsigned maxmq;
};
```

The **maxmsg** and **maxmq** fields of the dnet_ipcname structure are not currently used. The **name** field should contain the address (a simple character string) that your program wishes to have bound to its own endpoint, or of the endpoint of another program to which your program wishes to be connected.

The flags argument must have one and only one of the following flag values set:

D_CONNECT Find the IPC endpoint to which the address in **ipcname.name** is bound and connect to this endpoint.

D_BIND Bind the address in **ipcname.name** to this endpoint.

In addition, the D_GLOBAL flag may be set in combination with the D_BIND flag to force the address to be advertised globally across the current machine. This flag only has significance in the VMS environment since this is the norm in a UNIX environment. In order to have the address globally advertised, the process must have SYSNAM privilege.

The ipcget function returns an integer ipcid on success. This is similar to a file descriptor, but is not. Instead it is translated to a file descriptor, channel descriptor, or message type when used. This will be discussed more in the implementation section.

The following examples demonstrates the suggested usage of the ipcget library routine. The server program example is binding the address to its endpoint (thereby advertising the address), while the client program is connecting to the advertised address (at a time after the server has bound).

Server Program

```
#include "dnet_errno.h"
.
.
.
int ipcid;
struct dnet_ipcname ipcname;
.
.
.
strcpy(dnet_ipcname, "myaddress");
if((ipcid = ipcget(&dnet_ipcname, D_BIND | D_GLOBAL)) == -1)
{
        fprintf(stderr, "ipcget:dnet_errno(%d) errno(%d).0, dnet_errno, errno);
        return(-1);
}
```

Client Program

```
#include "dnet_errno.h"
.
.
.
int ipcid;
struct dnet_ipcname ipcname;
.
.
.
strcpy(dnet_ipcname, "myaddress");
if((ipcid = ipcget(&dnet_ipcname, D_BIND | D_GLOBAL)) == -1)
{
        fprintf(stderr, "ipcget:dnet_errno(%d) errno(%d).0, dnet_errno, errno);
        return(-1);
}
```

5.2.2.2 The ipcclose function

The ipcclose function frees resources associated with the IPC mechanism identified by the ipcid which is passed as an argument. On UNIX systems, this will also unlink the file entry in the DNET_IPCDIR.

5.2.3 Sending And Receiving Messages

Two routines are provided for sending and receiving messages over an established IPC mechanism. Validation is performed on each transaction to insure that the mechanism is capable of sending/receiving a message. Because the mechanisms are simplex in nature, the routines will not allow a message to be sent out an endpoint that has a bound address, and will not allow an attempt to read from an endpoint which is connected to a peer endpoint.

The descriptions of these two functions: **ipcsnd** and 3ipcrcv **follow:**

5.2.3.1 The ipcsnd function

The ipcsnd function allows a message to be sent out through and endpoint that has been successfully connected. The declaration of the ipcsnd function follows:

```
int ipcsnd(ipcid, umsg, umsglen, flag)
int ipcid;
char *umsg;
int umsglen;
int flag;
```

The **ipcid** argument is the endpoint identifier returned by the ipcget function. The **umsg** argument points to the buffer (binary data is acceptable) containing the data to be sent, while the **umsglen** indicates the number of bytes to be sent. The flag argument may have the D_NOWAIT flag set which will force the send to be non-blocking.

The following is a section of the same client program above demonstrating the use of the ipcsnd library routine. Because of the simplex connections, only the client program is allowed to use the ipcsnd routine on this IPC mechanism. The client program is not allowed to use the ipcrcv library routine on this IPC mechanism.

```
char umsg[D_MAXMSG];
int umsglen;
.
.
.
strcpy(umsg, "This does not have to be an ascii string");
umsglen = strlen(umsg) + 1;
if(ipcsnd(ipcid, umsg, umsglen, 0) == -1)
{
        fprintf(stderr, "ipcsnd:dnet_errno(%d) errno(%d).0, dnet_errno, errno);
        return(-1);
}
```

5.2.3.2 The ipcrcv function

The ipcrcv function allows a program to read a message from an endpoint that has an address bound to it. A description of the ipcrcv function follows:

```
int ipcrcv(ipcid, umsg, umsglen, flag)
int ipcid;
char *umsg;
int umsglen;
int flag;
```

The **ipcid** argument again identifies the endpoint over which the program wishes to receive a message. The **umsg** argument points to the buffer where the message will be placed, and the **umsglen** argument states how large that buffer is in bytes. The **flag** argument may have the D_NOWAIT flag set which will insure that the call does not block.

The following is a section of the server program above demonstrating the use of the ipcrcv library routine. Because of the simplex connections, only the server program is allowed to use the ipcrcv routine on this IPC mechanism. The server program is not allowed to use the ipcsnd library routine on this IPC mechanism.

```
char umsg[D_MAXMSG];
int umsglen;
int readlen;
.
.
.
umsglen = D_MAXMSG;
if((readlen = ipcrcv(ipcid, umsg, umsglen, 0)) == -1)
{
        fprintf(stderr, "ipcrcv:dnet_errno(%d) errno(%d).0, dnet_errno, errno);
        return(-1);
}
```

5.3 Implementation

This section will discuss the unique features of each operating system that were used to implement the standard IPC implementation.

5.3.1 The ipcid Table

The IPC module maintains a table of all active endpoints for a particular process. This table is very similar in function to the file descriptor table in UNIX operating systems. A short description of this table follows:

```
static struct
{
        char name[D_MAXFNAME];
        int flag;
        int id;
}ipctab[D_MAXIPCIDS];
```

The **name** field contains the address used in this IPC mechanism. The flag contains the flags specified on the ipcget, and the id contains a numeric value describing the lower level IPC mechanism. In System V environments this si a message type, in BSD environments it is a file descriptor for a socket, and in VMS environments it is a channel descriptor.

5.3.2 System V

The System V message queue facility was used to implement the IPC implementation on System V operating systems. This facility required work on three areas to bring it in line with the requirements of the IPC implementation:

- Standard interface
- Endpoint establishment
- Character string addresses

A description of the implementation of the standard interface follows:

_makeipc The ipc directory is created if requested and necessary. The message key value is used in attempt to create a new message queue for use by all processes using the to be created IPC medium. If the D_CREAT flag is not set, then an attempt will be make to look up an existing message queue with a matching key value, and will fail if one does not exist. If the D_CREAT flag is set, then the queue will be created if it cannot be found. If the D_CREAT and D_EXCL flags are set, then the call will only succeed if a message queue with the requested key value did not previously exist. The flag values are used in a similar nature for the creation of the ipc directory.

_removeipc If it is determined that this program called _makeipc with both the D_CREAT and D_EXCL flags set, then the ipc directory will be removed, and the message queue will be freed and returned to the system. In all other cases the call always returns successfully.

ipcget The ipcget routine attempts to find a file in the ipc directory with the same name specified as the address requested in the ipname structure. If the D_CONNECT flag is set and the file exists, then a message type value is determined (as described below) and is placed in the id field of the appropriate entry in the ipcid table.

If the D_BIND flag is set, then a file is created in the ipc directory, a message type value determined and is placed in the id field of the appropriate entry in the ipcid table.

ipcclose The ipcclose routine will remove the file from the ipc directory as long as the D_BIND flag was used on the ipcget. If the D_CONNECT flag was used, then the file will remain.

ipcsnd The ipcsnd routine packages the message into a System V message queue structure, sets the message type field to be that of the id field in the ipcid table and adds the message to that queue.

ipcrcv The ipcrcv routine attempts to read a message from the queue where the message type matches the id field in the ipcid table.

Making the midpoint characteristics of the message queues emulate endpoint characteristic was accomplished by creating file nodes in the ipc directory for every IPC mechanism (note that this is for every mechanism and not for every endpoint). This allows a simple check to be done to insure that before an attempt to bind is made, that there is not another process bound to that address, and that before an attempt to connect is made, that another process has bound to that address and is ready to receive.

Mapping a character string name to the message type value was performed by merely obtaining the inode number of the file node created for the IPC mechanism. Because all file nodes are created on the same file system (they are all in the same directory), the inode number is unique. In addition, the inode number is the same for every process that checks it and so provides a stable conversion.

5.3.3 BSD

The BSD socket interface was used with the UNIX address family as the underlying mechanism of IPC in Berkeley UNIX systems. This facility required work only in the area of interface to bring it in line with the requirements of the IPC implementation. One apparent bug in the operating system makes the IPC mechanisms system hogs during excessive use of the IPC mechanisms. This is discussed in the description of the ipcsnd interface.

A description of the implementation of the standard interface follows:

_makeipc The _makeipc routine creates/references the ipc directory in a fashion identical to that of the System V _makeipc.

_removeipc The _removeipc routine acts identical to the _removeipc routine of System V excluding the freeing up of the System V message queue.

ipcget The ipcget routine translates almost directly into a socket system call followed by a bind system call if the D_BIND flag is set or a connect system call if the

D_CONNECT flag is set. The file descriptor returned by the socket system call is placed in the id field of the appropriate record in the ipcid table.

ipcclose — The ipcclose routines uses the close system call on the file descriptor in the ipcid table, and then, if the D_BIND flag was specified on the ipcget, then the file node is removed explicately from the ipc directory. The BSD system does not yet remove the file nodes it creates on the bind system call.

ipcsnd — The ipcsnd maps almost directly to the send system call. There exists a bug, though, in the BSD implementation of the IPC on send where, even when blocking mode is set (this is by default), the system will return with a E_NOBUFS error when there is a transient shortage of buffers in the system to place the message. With that attitude that this bug will be fixed, the ipcsnd routine loops (eating up valuable CPU resources) until the message can be taken or a more definitive error occurs.

ipcrcv — The ipcrcv routine maps directly to the recv system call.

5.3.4 VMS

The VMS mailbox interface was used as the underlying mechanism of IPC in VMS systems. This facility required work in the following areas to bring it in line with the requirements of the IPC implementation. The VMS system, in addition actually fails implied requirements of the IPC implementation in that SYSNAM privilege is required to advertise globally. To partially overcome this, the ipcget routine in VMS returns the actual device name of the mailbox accessed by the ipcget call. This name may be passed, through some means, to the person attempting to connect to your endpoint. The problem with this is that the fully qualified mailbox name can be very long (especially in cluster environments). This required that all IPC implementations increase their overhead to accommodate for the extra space required by VMS. This can be overcome, but lack of time and resources limit us at this time.

- Standard interface
- Endpoint establishment
- Independence between peers

A description of the implementation of the standard interface follows:

_makeipc — This is an effective noop function call in VMS.

_removeipc — This is an effective noop function call in VMS.

ipcget — This is probably the most complex of all the IPC routines because of all the facilities potentially touched. In general, a channel is assigned to the address specified in the name field of the ipcname structure if the D_CONNECT flag is set. The name field is either a logical name which will translate to a mailbox device name, or is the direct mailbox device name itself. If the logical name cannot be translated, and error is returned indicating that no peer exists.

If the D_BIND flag is set a mailbox is created. A logical name is equated to this mailbox, and a channel is assigned to the logical name. This results in the logical name being placed in the job table. If the D_GLOBAL flag was set, then an attempt is made to assign the same logical name to the system table. In both events for the bind, the device name of the mailbox is copied over top of the logical name in the

name field of the dnet_ipcname structure. The channel number is placed in the id field of the appropriate ipcid table entry.

ipcclose If this endpoint had an address bound to it, then the logical name entries are removed from all appropriate tables, and the mailbox device is freed for use elsewhere.

ipcsnd The ipcsnd routine is implemented with the standard SYS$QIOW system service. The VMS mailbox facility will normally attempt to hold the write outstanding until a peer has attempted to read it. The IO$M_NOW flag was set to force the write into the mailbox and prevent it from remaining outstanding.

ipcrcv The ipcrcv routine is also implemented using the standard SYS$QIOW system service.

The requirement of endpoint establishment is not met under VMS. Two sticking points still exist: 1) The global advertisement of addresses requires SYSNAM privilege (which is an unfeasible expectation) and therefor opens the door to multiple processes binding to the same name. In addition, the logical tables are allowed to be overwritten with new values, meaning that no check is performed to see if the name has already been bound to. The dnet services currently compensate for this under VMS environments.

The requirement of peer independence was met through a combination of the IO$M_NOW flag and the SYS$SETRWM system service. The IO$M_NOW flag was set in the ipcrcv routine to initiate a non-blocking read. In the ipcsnd routine, the IO$M_NOW flag must always be set, and the SYS$SETRWM system service was used to temporarily set the resource wait mode from its default of waiting for the resource to the state in which it will fail if the resource is not available (a full mailbox in this case).

6. Miscellaneous DNET Internal Utilities

This section describes miscellaneous utilities which are internal to DNET.

6.1 General System Utilities

6.1.1 getppid
6.1.2 fperror
6.1.3 iosync
6.1.4 is_error
6.1.5 prttime
6.1.6 stricmp

6.2 General Network Utilites

6.2.1 check_mynet
6.2.2 disassemble
6.2.3 dn_init
6.2.4 dn_makedg
6.2.5 dn_makepvc

6.3 Stream to Datagram Conversion Utilities

6.3.1 strtodg_dglen
6.3.2 strtodg_msg
6.3.3 strtodg_numhops
6.3.4 strtodg_path
6.3.5 strtodg_pathlen
6.3.6 strtodg_stream
6.3.7 strtodg_stream_msg
6.3.8 strtodg_type

6.4 UNIX Specific Utilities

6.4.1 build_argarr

6.4.2 execshell
6.4.3 startserver

6.5 VMS Specific Utilities

6.5.1 create_mailbox
6.5.2 execshell
6.5.3 getargs
6.5.4 gobetween
6.5.5 setargs
6.5.6 startserver
6.5.7 lib_do_command
6.5.8 lib_spawn
6.5.9 sys_assign
6.5.10 sys_cancel
6.5.11 sys_crelnm
6.5.12 sys_crelnt
6.5.13 sys_crembx
6.5.14 sys_creprc
6.5.15 sys_dassgn
6.5.16 sys_dellnm
6.5.17 sys_delmbx
6.5.18 sys_getdvi
6.5.19 sys_getjpi
6.5.20 sys_getmsg
6.5.21 sys_hiber
6.5.22 sys_qio
6.5.23 sys_qiow
6.5.24 sys_trnlnm
6.5.25 sys_wake
6.5.26 vms_fperror
6.5.27 vms_perror
6.5.28 vms_read
6.5.29 vms_write

6.6 MS DOS Specific Utilites

To be added

7. Interfaces to Underlying Networks

Both the Datagram Assembler/Disassembler and the Router of the BASIC I/O Package connect to underlying networks via the Network I/O Interface. This interface "maps" generic function calls (dn_open, dn_close, dn_read, dn_write, etc.) into protocol specific functions for a particular network.

The files **tcp.c and decnet_nt.c** in the **network specific interfaces for most TCP or DECnet systems. The files exostcp.c contain**

7.1 Underlying Network Protocols

Wherever possible, existing, well known network protocols are employed in order to achieve reliable communication services between DNET nodes. These protocols are internally sophisticated, typically containing their own queing, buffering, retry and timeout mechanisms as well as their own routing **within their own network domain**. Despite this internal complexity, it is important to note the following:

- **From the DNET perspective the protocols provide point to point link and physical level services between nodes defined in the the DNET network.**

For each protocol the following generic functions are provided:

- Open
- Init_Permanent_Server
- Init_Transient_Server
- Get_Client
- Close
- Read
- Write
- Async_Read
- Wait

Two protocols are currently supported within DNET. These are:

1. TCP/IP
2. DECnet

The specific interfaces to these protocols are discussed in the following sections:

7.2 TCP/IP

Three implementations of TCP/IP are in use within DNET. The usage varies with the particular DNET node. The three implementations of TCP/IP currently supported together with the relevant

host machines are:

1. Berkeley UNIX - DAC & NASA Sun's
2. Wollongong - DAC MicroVAX II and 3B2/600
3. Excelan - NASA VAX's

Common source code is used for all three implementations. This code is located in the file **tcp.c**.

7.3 TCP/IP Specific Utilities

The following 'tcp/ip' specific functions are supported by DNET:

7.3.1 tcp_accept
7.3.2 tcp_close
7.3.3 tcp_getclient
7.3.4 tcp_initperm
7.3.5 tcp_inittrans
7.3.6 tcp_open
7.3.7 tcp_pvcopen
7.3.8 tcp_read
7.3.9 tcp_write

The reader is referred to the source listings for **tcp.c** for further details on these functions.

7.4 DECnet

Source code for the DNET interface to DECnet is found in the source files **decnet.c and decnet_nt.c**. The supported functions include:

7.4.1 _decnet_read
7.4.2 decnet_accept
7.4.3 decnet_close
7.4.4 decnet_errgeneric
7.4.5 decnet_errprotocol
7.4.6 decnet_getclient
7.4.7 decnet_initperm
7.4.8 decnet_inittrans
7.4.9 decnet_open
7.4.10 decnet_pvcopen
7.4.11 decnet_read
7.4.12 decnet_select

7.4.13 decnet_write
7.4.14 vms_aread
7.4.15 vms_awrite
7.4.16 vms_wait

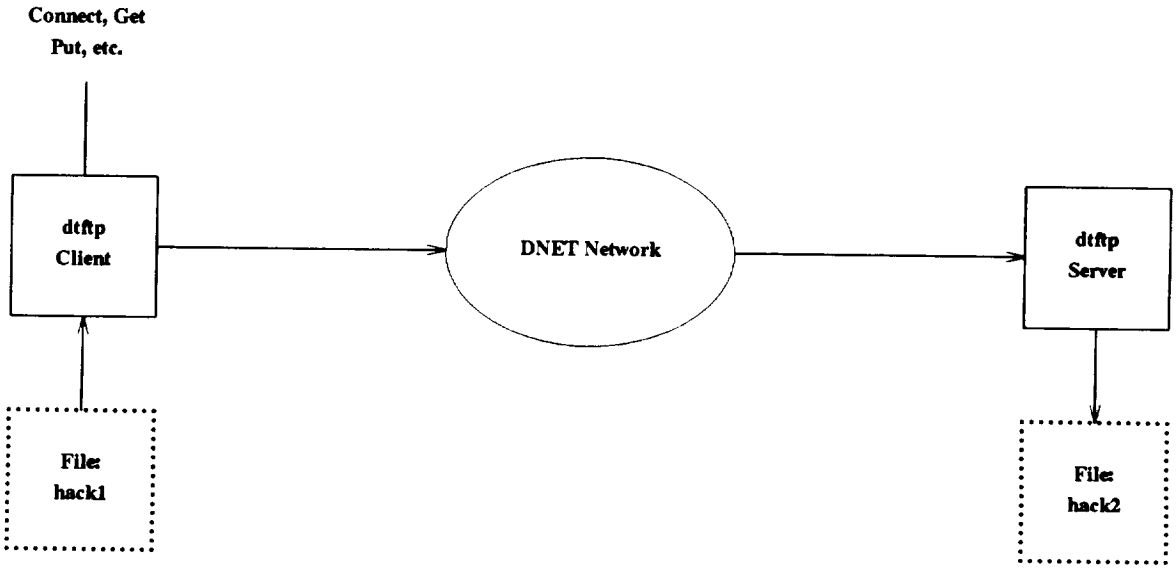
8. User Application Internals

8.1 File Transfer Protocol

The DNET File Transfer protocol

dtftp transfers blocks in fixed size (512 byte) units. Acknowlegements are sent by the receiving host's file transfer server (dtftpd) after each block has been received. Error reporting packets include the following:

8.2 Schematic of File Transfer



8.2.1 General Considerations

The receiving host tests for existance of the target file using the "access" function and gives notice if the file exists and creates a new version (if version numbers are supported by the local file system). Default values for protection mode and sharing options are used.

8.2.2 ASCII

The routines aput() are used to transmit 'text' or ASCII format files. The 'formatted' i/o calls fopen, gets, etc. are used for file access in this mode.

8.2.3 Binary Files

8.3 Security During File Transfer

When the client invokes **dtftp**, authentication of the client is done by the login process at the remote host. Subsequent process spawning and/or remote login to other hosts from processes created by the initial client will all carry the access rights permitted to the initial client.

8.4 Initiation of File Transfer from One Remote Node to Another

The Network Command Language may be used at a third party location to initiate file transfer. A typical command would be:

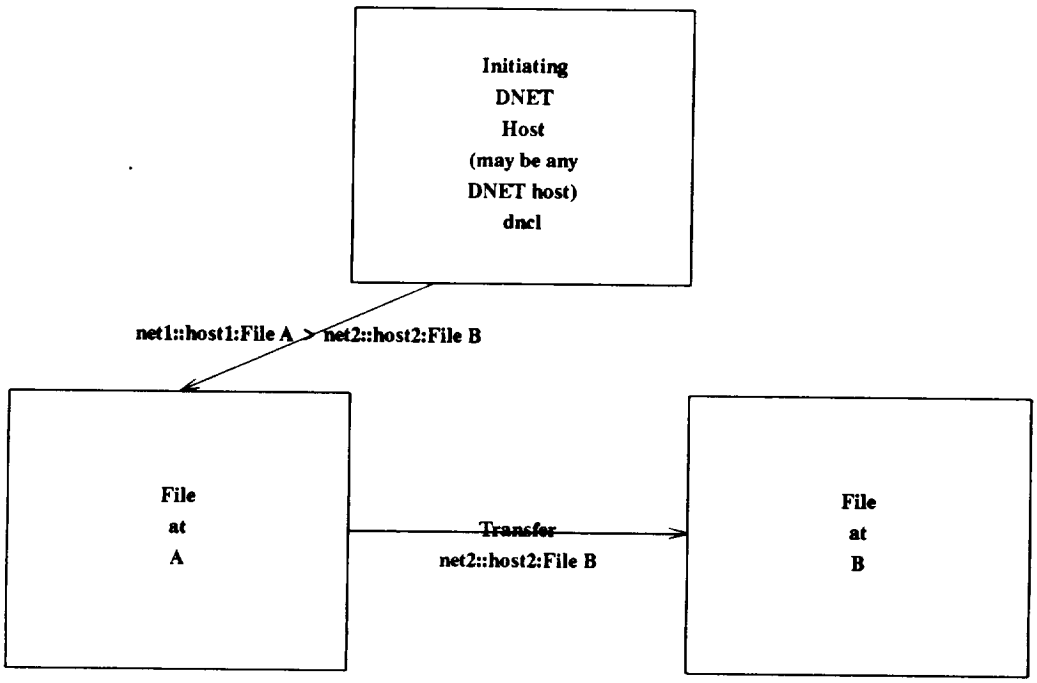
dncl > net10::host3:filexx > c-net::fhost:newfile

or

dncl > mynet::host6:*dtftp filename options > > newfile

Where filename and options are parameters to the file transfer task "**dtftp**".

The effect of such a command is shown in the following diagram:



8.5 Initiation of Remote Procedure Upon Completion of File Transfer

It is also possible to use the DNET Network Command Language to perform a file transfer followed by the execution of a remote procedure. Several alternatives are possible.

1. Two separate commands:

   transfer the file

   dncl > bnet::host3:file4 > c-net::xhost:newfile

   followed by

   execute the remote procedure

   dncl > c-net::xhost:*format newfile

2. **One 'composite' command:**

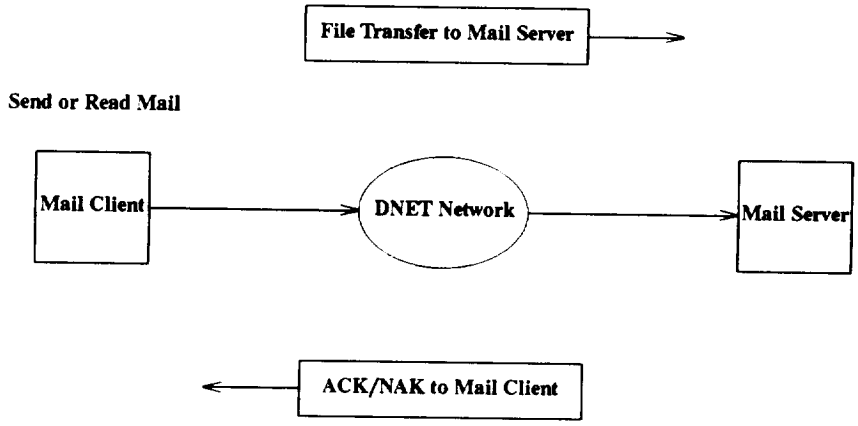
   dncl > bnet::host3:file4 > c-net::xhost:newfile | c-net::xhost:*format

8.6 Remote Login

8.7 Electronic Mail

8.8 General

DNET provides a very basic Electronic Mail facility.



8.9 Mail Operation

8.9.1 Structure of DENT mail files

The organization of DNET mail files is as follows:

8.9.2 Sending Mail

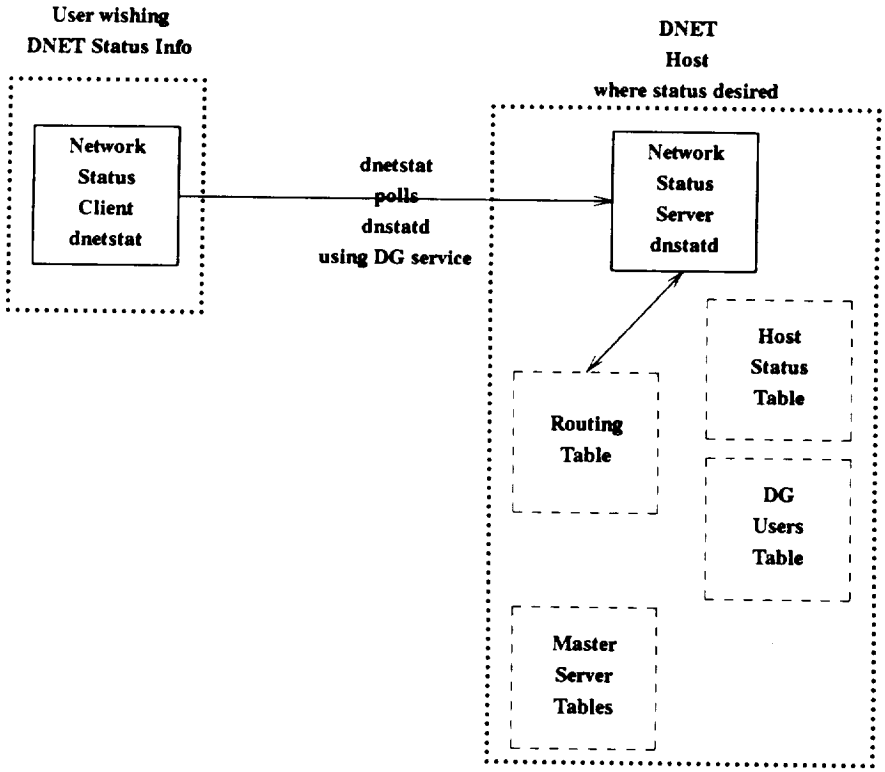
8.9.3 Reading Mail

8.9.4 Mail Routing

Routing of mail is implicit. The user sending mail must know the (DNET) destination host, network and user account name of the receiving party.

9. dnetstat - Network Status Function

The status of Master Servers and the servers they spawn will be monitored by a program operating on one or more of the network hosts. Status of the Master Servers on the local network will be obtained using the facilities provided by the networking software native to the local network. Status of the servers created by the Master Servers can be obtained in the same way because the names of these processes can be derived from their parent.



10. Network Command Execution & Task Redirection

The Network Command Processor is a command language processor for use in a heterogeneous multi-network environment. A terminal user interface and a "C" language interface to this processor will be provided.

This DNET facility allows very general control of processes across the heterogeneous network and provides for redirection of input/output streams between files and/or processes located at arbitrary DNET Hosts.

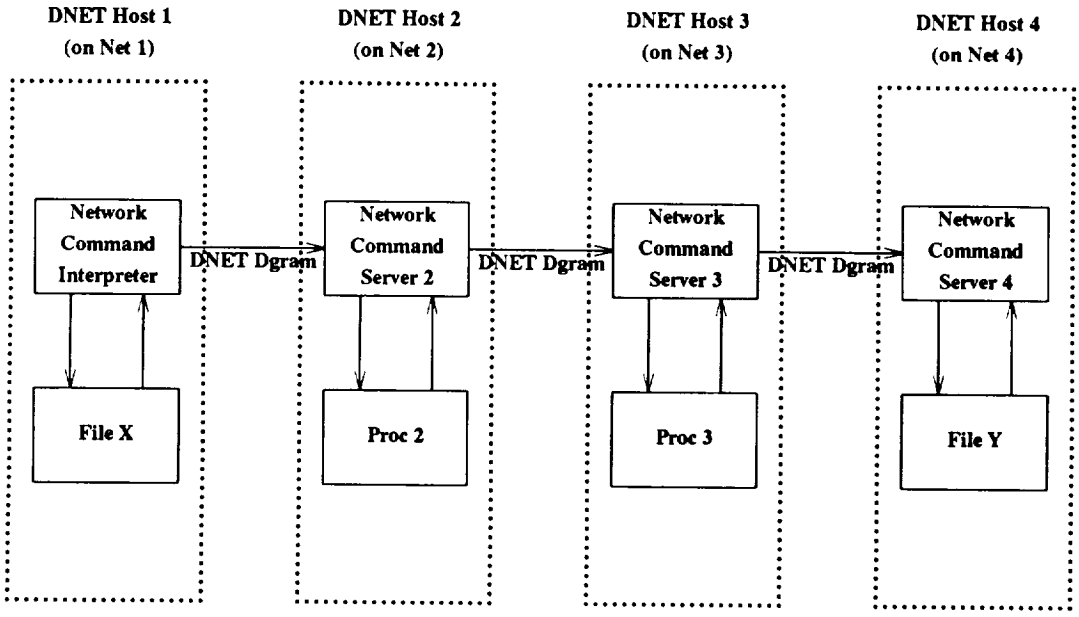
Three Software Elements are required for the Network Command Processor.

- Network Command Interpreter (NCI) - used at the initiating node to interpret the Command Line (CL) entered by the user, divide this CL into separate Sub Command Lines (SCL) and pass these on to the first NCS.
- Network Command Server (NCS) - services network command which arrives from NCI or another NCS; provides any local service requested, including process spawning, and sends remainder of the SCL to the next NCS in the command chain The NCS is a DNET application server and is thus registered in relevant domain server tables.

A Schematic view of the relationship between these components is shown in the figure below. The generic command string being executed is:

Net1::Host1:File X > Net2::Host2:*Proc2 > Net3::Host3:*Proc3 > Net4::Host4:File Y

10.1 Network Command Processor Schematic



10.2 Network Command Language

10.2.1 Command Language Syntax

There are two types of objects- Files and Filters. The ">" operator is used to delimit the SCL components of the CL.

Filename syntax is: network_name::host_name:filename

Taskname syntax is: network_name::host_name:*taskname(param1, param2 ...)

An example command is:

starnet::xhost:cfile > yhost:*sort > myfile

Other examples are given below.

When the network name or host name is not specified the local name is assumed. Spaces around the ">" are optional.

10.2.2 Using The Command Language

When filenames appear in command strings they imply the execution of file i/o servers. The network command:

dac_net::vax2:david.comm > g_net::host1:*checkp > results

requests that the contents of a file "david.comm" on host "vax2" in the network "dac_net" be run through the filter "checkp" executed on "host1" in the network "g_net, and the output be placed in the file "results" in the host on which the previous NCS was run (g_net::host1 in this case).

The network command:

net_one::vax6:c-file > host1:s-cfile

requests that the contents of a file "c-file" on host "vax6" in the network "net_one" be copied to the file "s-cfile" on the host1 machine on the same network. The network command:

10.3 Network Command Interpreter

The Network Command Interpreter is invoked as an application from the shell prompt on the local system.

%) dncl

%) dnc

dncl> command_string1

response to CS1

dncl> command_string2

response

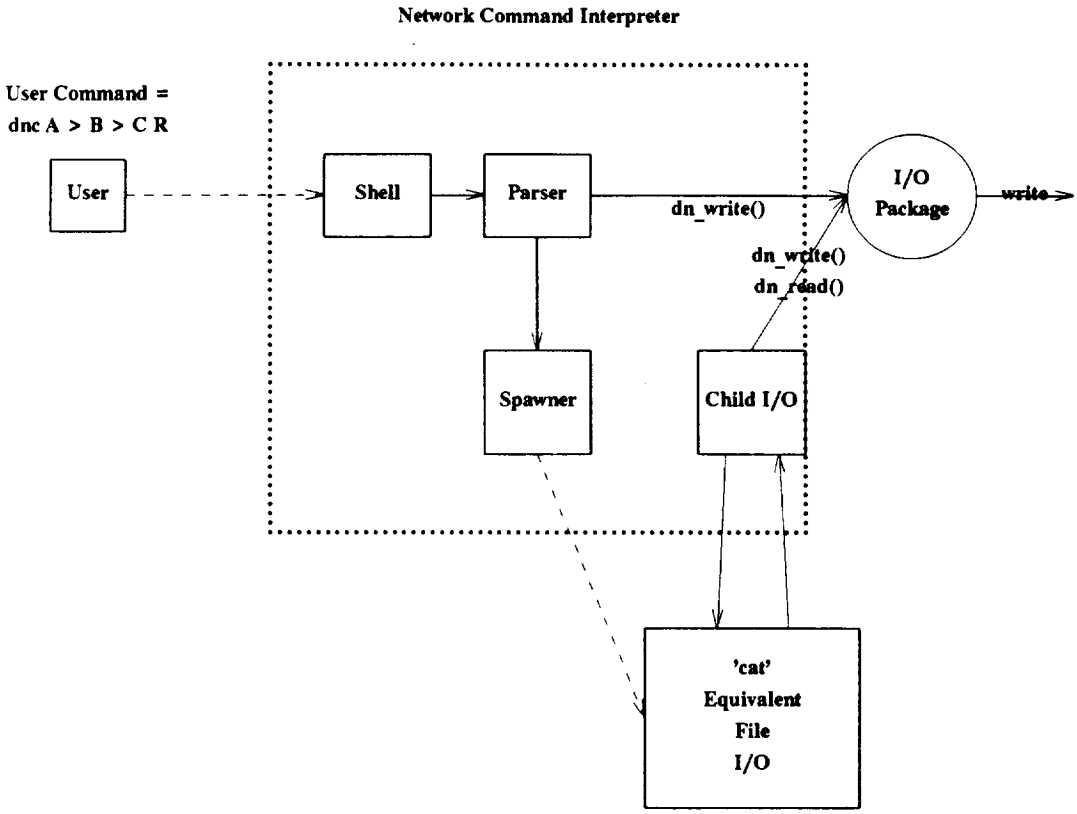
etc.

After parsing the CL, the NCI module opens a dnet connection to the NCS module specified in the first SCL. The complete list of SCLs are passed over this connection to the NCS component.

All interaction between NCS components and between the NCI and NCS components are via standardized packets. The packet header contains a length field and packet type field to describe the data (if any) that follows.

The NCI module then wait for an ACKCOMP packet type to be recieved over the connection just established to send the CL to the first NCS module. An ERROR packet type may also be received at this point, and the data within the packet would be an error message generated at one of the downline NCS modules.

10.3.1 Schematic of Network Command Interpreter



10.4 Network Command Server

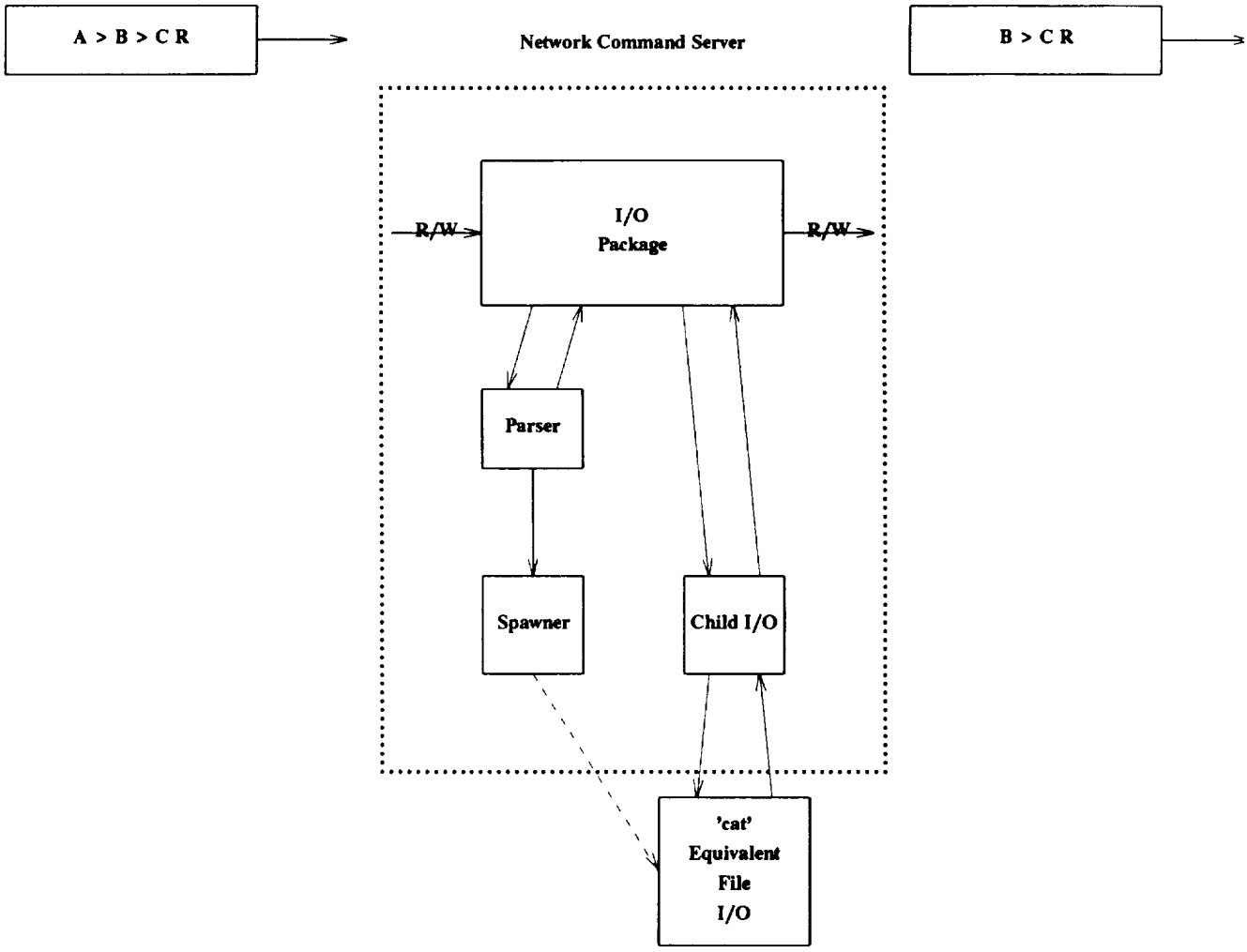
The NCS module is set up to provide for exactly one SCL. This may involve reading a file, spawning a filter, or creating a new output file. After the last NCS module has completed successfully, it will initiate an ACKCOMP packet to inform all NCS modules upline, and the initial NCI module that the operation was completed successfully.

Operations at the NCS include:

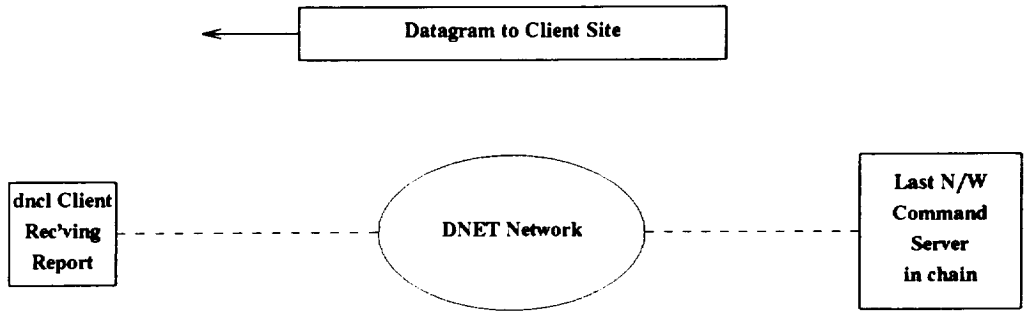
1. Wait for the NCI client or an upline NCS component to request a connection.
2. Read the CL (one SCL at a time) from the established dnet connection.
3. Determine SCL category
   - First SCL on the CL -- read file
   - A middle SCL -- filter
   - Last SCL on the CL -- create output file, initiate ACKCOMP
4. Read input data packets until EOF packet arrives
5. Send EOF packet downline
6. Wait for ACKCOMP packet to arrive from downline
7. Send ACKCOMP packet upline

8. Close upline and downline channels

10.4.1 Operations at Network Command Server during File I/O



10.4.2 Status Reporting (from last Network Command Server)



10.5 An Example

An example command is:

net1::host5:*lookup< > net2::host4:*sort > net5::host1:fllex

The execution of this command is discussed below.

To support the execution of network commands two types of tasks are used: network command servers (net_com_serv) and network i/o servers (net_file_io).

The net_com_serv tasks will assist in the remote execution of network commands by accepting messages from other hosts' network command servers, spawning tasks as required, reading from and writing to other hosts, passing the data to the spawned task as "standard input"(SYS$INPUT) and taking "standard output"(SYS$OUTPUT), thereby allowing the spawned tasks to operate in the network environment without modification.

The second type of supporting task is the net_file_io. It is used to transmit and receive files. When a filename appears as the only object in command component (i.e. it is not a parameter to a task), it is assumed that the task to be executed is net_file_io.

The procedure used by a network command server to execute a network command is:

1. Read a command line from a network command language processor, or a network command server
2. After deleting that portion of the command that is being executed by the current host, send a copy of the command line to the host that will execute the next part of the command line
3. Identify the first task name in the command line (scan from left)
4. Spawn the identified task using the host that sent the command line in step 1. as the source of standard input to the spawned task (pass data through a mail box to the spawned task) and send the output from that task to the network command server on the host which will execute the next task in the command line.

When the network command server controlling the last task in the command string completes, it sends

a termination message, with status information, to the network/host/process that initiated the command execution chain.

10.6 An Example of Network Command Execution

As described above, the network commands will be processed by distributing all or part of the command line to various hosts for execution. Processing will start by sending a copy of the command line from the network command language processor to the network command server on the system which will execute the first task in the command line. To execute the network command:

net1::host5:*lookup< > net2::host4:*sort > net5::host1:filex

the following processing is performed:

1. The network command language processor sends a copy of the command line to net1 host5
2. The network command server on net1 host5 will send to net2 host4 a copy of the command line text starting at "net2::host4".
3. Then identify the task "lookup" as the task to be executed.
4. Spawn a copy of the lookup task with standard input comming from the host that sent the command line and standard output going to net 2 host 4. Both standard input and output streams pass through the mailbox shared by the lookup task and the network command server which spawned it.
5. Status messages are sent to the network command processor that invoked this command execution.
6. The network command server on net2 host4 will send to net5 host1 a copy of the command line text starting at "net5::host1",then
7. identify the task "sort" as the one to be executed.
8. Spawn a copy of the sort task with standard input being from the host which sent the command line and standard output to net 5 host 1. Both standard input and output streams pass through the mailbox shared by the sort task and the network command server which spawned it.
9. The network command server on net 5 host 1 will read the command line and
10. identify "filex" as a filename, therefore choose task "net_file_io" as the one to be executed.
11. Since there is no more text in the command line there is no successor task to send the command line to. The network command server spawns the net_file_io task with "filex" as the output file and standard input being from the host which sent the command. Data from the input source is read and stored in the output file until an end of file causes the termination of the net_file_io task.
12. This causes the parent process, "net_com_serv" to send a completion status message to the process that initiated the execution of this command string.

10.7 Network Command Processor Implementation

There are three major components in the Network Command Processor:

- Network Command Interpreter
- Network Command Server
- Network File I/O

The implementation of these is outlined below.

10.8 Network Command Interpreter

The Network Command Interpreter reads commands from users, parses, processes, and distributes them to the network servers on the specified hosts. messages sent to the servers that execute the command.

In the design presented below the symbol used for the data flow operations is ">" .

"Names" refer to either files or tasks (the "*" precedes tasknames).

The datagrams sent to the Network Servers are produced as follows:

1. Pointer P1 is set to the start of the first name in the command.
2. Starting at P1 text is scanned, stopping at the first op code it finds, or the end of the command line, whichever is found first. If it finds the end of the command line a flag is set (see below for processing done for this).
3. The op code is saved in 'op'.
4. P2 is set to the start of the name following the op code.
5. Scan as in Step 2 to the next op code.
6. Using P1 'op' and P2 genetate the skeleton form of the message that will be sent to the host whose name is pointed to by P1.
7. Set P1 to the value in P2.
8. If the "end" flag is not set go to Step 2.

10.8.1 Additional Processing

Additional processing of messages is required to add information about "implied" servers and parameters for file names and for the return of completion status. Samples of the types of messages that require this processing are shown below.

Original Message	Modified Message
Host1:*task1 < > Host2:*task2 or Host1:*task1 > Host2:*task2	Prefix message with name of net_com_serv.
Host1:*task1 > Host2:file2	Prefix message with name of net_com_serv. Replace file2 with *net_file_io,(create, file2)

Host1:file1 > > Host2:file2	Prefix message with name of net_com_serv. Replace file2 with *net_file_io,(append, file2)

In addition to the modifications shown above, each message will be given a serial number uniquely identifying the command with which it is associated and the network address of the command interpreter to which completion status will be sent.

10.9 Network Command Server

The Network Command Interpreter sends messages to the various hosts specified in a network command. The messages contain the name of a server, the parameters to be used in processing, and the network address of the Network Command Server to which the completion status code should be sent.

10.9.1 Implementation of the Network Command Server

The Network Command Server listens for datagrams from any Network Command Interpreter. Upon receiving one it determines the name of the server being requested, the parameters to be used in the call to it, and the network address of the Network Command Interpreter that sent this request.

On VMS systems the following is done:

To obtain the name of the mailbox associated with an instance of the requested server the local Master Server is called. The Network Command Server then writes a message to the mailbox, requesting a local service connection. In this mode of operation the client (Network Command Server) provides the names of one input and one output mailbox to be used by the requested server for SYS$INPUT and SYS$OUTPUT. During the execution of the command the Network Command Server continuously reads from the network connection to the prior host in the command pipeline and writes to the SYS$INPUT mailbox. At the same time it continuously reads from the SYS$OUTPUT mailbox and writes to the network host on which the next task in the command pipeline is executed.

After the completion of the command execution the Network Command Server Deassigns the mailboxes used in the command, keeping them for future use. The completion status code is returned to the originating Network Command Interpreter by sending a datagram.

On UNIX systems the processing is as follows:

Each server is created by the Network Command Server when needed, using the fork and exec system calls. In this way the standard input/output files, in this case pipes, created by the parent (the Network Command Server) are available to both parent and child. During the execution of the command the Network Command Server reads from the network connection to the prior host in the command pipeline and writes to the standard input of the child. At the same time it reads from the standard output of the child and writes to the network host on which the next task in the command pipeline is executed.

The completion status code is returned to the originating Network Command Interpreter by sending a datagram.

10.10 Network File I/O

Network File I/O is a server that is used to read and write files on the local host to and from remote hosts. It assists in transmitting input and output data across network connections that support

command pipes.

The arguments to Network File I/O are

- mode (append or create)
- filename

11. Presentation Layer Services

11.1 XDR

dn_xdr.c

12. DNET Error Handling

DNET Basic I/O Library functions return a non-selective error code if an error is detected during their operation. These errors are defined in the header file **../dnet/common/dnet_errno.h**

Errors detected by the DNET code are identified in the variable **dnet_errno**:

```
dnet_errno = XXXXX;
```

An error function, **dnet_error("string")**, is then optionally called where string is an optional, user provided informative message. **dnet_error** provides detailed information on conditions when the error was detected including a stack trace.

```
dnet_error(*error_string)

char * error_string;
```

Detailed error codes are provided in the programmer reference manual.

13. Routing

13.0.1 get_path

13.0.2 load_my_name

13.0.3 load_net_table

The router selects the host/process to which the datagram will be transmitted next.

get_path();

path = get_path(src_net,src_host,dest_net,dest_host,dest_process,numhops);

src_net is the network in which the destination host is located

src_host is the destination host

dest_net is the network in which the destination host is located

dest_host is the destination host

dest_process is the destination process

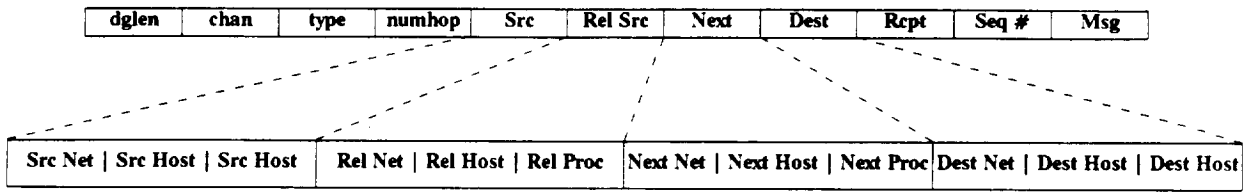
numhops - number of hops from current location to destination

13.1 Router Operation

The paths to hosts in the local network are direct connections. For paths to hosts in other networks a dynamic router is used. A hierarchical routing table is used to determine the host to which the datagram should be sent next. The entries in the routing table are updated by exchange of connectionless datagrams between DNET gateways and individual DNET hosts.

In the future the router may be enhanced to include searching for alternate paths and servers if the standard search fails to satisfy the request. The second search could extend into other networks in requests for generic servers that need not be executed in a specific network or host. Extended searches will provide automatic alternate routing, load sharing, and backup services for use when failures in hardware or software reduce the availability of facilities.

The datagram header contains three fields which are used in routing as indicated below:


dglen
chan
type
numhop
Src
Rel Src
Next
Dest
Rcpt
Seq #
Msg
Src Net | Src Host | Src Host
Rel Net | Rel Host | Rel Proc
Next Net | Next Host | Next Proc
Dest Net | Dest Host | Dest Host

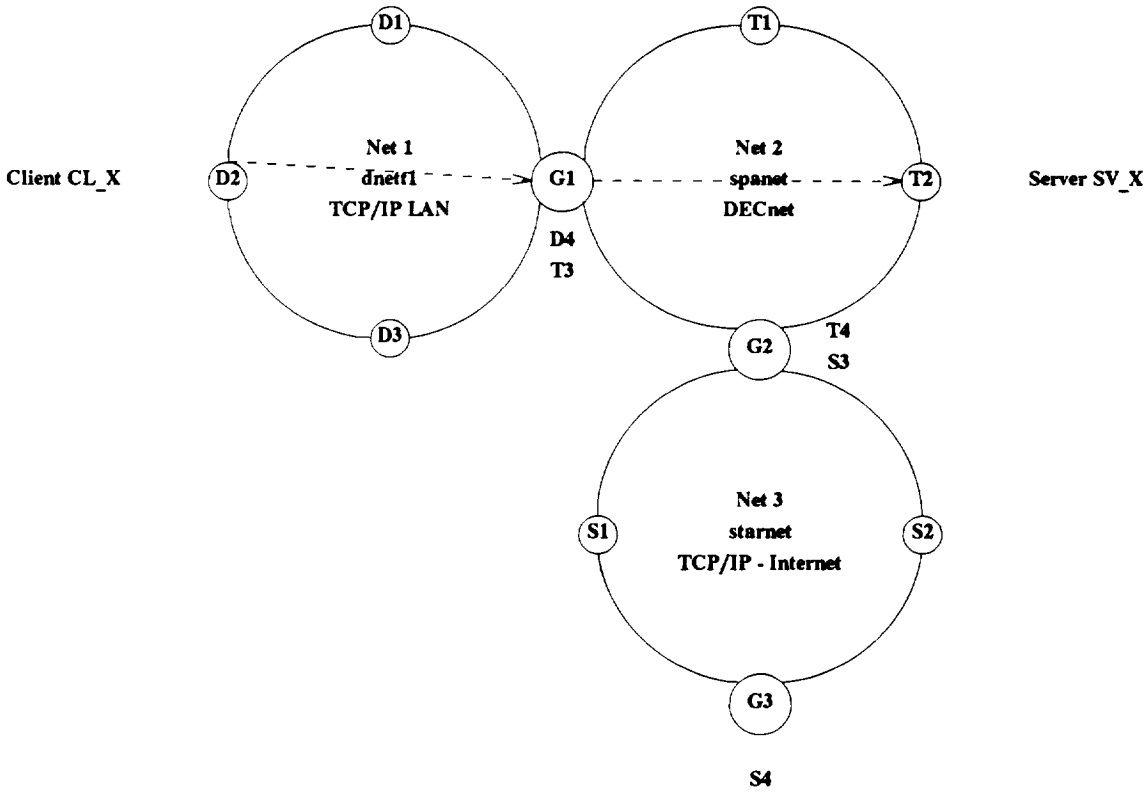
```
typedef struct {
  short dglen;    /* The total length of the datagram, excluding
                  this field */
  short chan;     /* The channel number that is being used */
  short type;     /* A code for the datagram type - Connectionless,
                  Virtual Circuit or Signal */
  short hopnum;   /* The curent hop number-- to catch circular routing*/

  char  srcnet;   /* The DNET code for the src host's network name */
  short srchost;  /* The DNET code for the src host's host name */
  char  *srcproc; /* The name of the process to be used on the src host */

  char  rel_srcnet;   /* The DNET code for relative
                      src host's network name */
  short rel_srchost;  /* The DNET code for the src host's host name */
  char  *rel_srcproc; /* The name of process to be used on the src host */

  char nextnet;       /* Next DNET network to be reached */
  short nexthost;     /* Next DNET host (on nextnet) */
  char *nextproc;     /* Process to be contacted on 'nexthost' */

  char  destnet;  /* The DNET code for the dest host's network name */
  short desthost; /* The DNET code for the dest host's host name */
  char  *destproc; /* The name of the process to be used on the dest host */
  char receipt;   /* Return Receipt Request = 0 no receipt
                              1 receipt requested

  char *sequence#         /* PID and datagram sequence number */

  char  *msg;     /* The data to be sent */
} DATAGRAM;
```

A typical routing table is shown below:

DNET Local Routing Table			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
dnett1	-	-	udp
spanet	dacvax	drelaytd	udp
starnet	dacvax	drelaytd	udp
Net X	Host Y	drelaytX	udp
-	-	-	-

13.2 Routing Example

The route generated for a typical datagram is shown in the following diagram:



In this example client CL_X on DNET host D2 wishes to conduct a session with server SV_X on DNET host T2.

The router on host D2 has the following routing table available:

DNET Local Routing Table - Host D2			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
dnett1	NULL	NULL	udp
spanet	dacvax	drelaytd	udp
starnet	dacvax	drelaytd	udp
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

The router on host D4 has the following routing table available:

DNET Local Routing Table - (Gateway) Host D4			
Destination Net	**Next (Gateway) Host**	**Next Process**	**Datagram Protocol**
spanet	NULL	NULL	**dec**
dnett1	**dacvax**	**drelaytd**	**udp**
starnet	**iaf**	**delaydt**	**dec**
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

13.3 Routing Table Updates

Initially, routing table updates will be handled in a manual fashion. Examination of a method for automatic updates for these tables will be considered as time allows.

DNET

TECHNICAL REFERENCE

Version: 1.10
Print Date: 08/31/89 17:37:38
Module Name: tech.ref

Digital Analysis Corporation
1889 Preston White Drive
Reston, Virginia 22091
(703) 476-5900

NAME

ass_dg - assemble a dnet datagram.

SYNOPSIS

```
#include "dnet"

int ass_dg(udg, ddg)
struct udg *udg;
char *ddg;
```

DESCRIPTION

The ass_dg internal library routine takes the contents of the udg structure and assembles a standard dnet datagram into the ddg buffer.

This function is used for purposes of preparing the user datagram to go over a network. Integer conversions are performed here as necessary. This function is only called by the per protocol dgs components.

SEE ALSO

dass_dg(3I)

RETURN VALUE

The ass_dg routine will return the size in bytes of the assembled datagram if successful. If an error condition exists, then the return value will be -1 and the external variable dnet_errno will hold the error value.

ERRORS

The call will not currently return in error.

NAME

check_mynet - validate the name of default network

SYNOPSIS

#include "dnet.h"

int check_mynet()

DESCRIPTION

This routine checks the name of the default network (retrieved by load_my_nmae(3I)) against entries in the tbls.net table (loaded by load_net_table(3I)) to insure the the default network is truly defined.

This routine is currently only called from the dn_init(3U) routine.

SEE ALSO

dn_init(3U), load_my_name(3I), load_net_table(3I)

RETURN VALUE

The routine returns a value of zero on success, and -1 to indicate an error.

ERRORS

The call fails if:

[D_INTERN] The default network name could not be found in the tbls.net table. This would indicate an administrative error.

NAME

dass_dg - dissasemble a received dnet datagram.

SYNOPSIS

```
#include "dnet.h"

int dass_dg(ddg, udg)
struct udgbuf *ddg;
struct udg *udg;
```

DESCRIPTION

This routine dissasembles a datagram received from the network into the structure used by dnet user programs and the dgms.

The per protocol dgs components are the only components that need to call this routine. Network integer conversions are performed for the header information in this routine as needed.

SEE ALSO

ass_dg(3I)

RETURN VALUE

The routine will return a value of 0 on success and a value of -1 to indicate an error condition.

ERRORS

This routine will not currently return in error.

NAME

dbcopy - binary copy

SYNOPSIS

```
int dbcopy(frombuf, tobuf, len)
char *frombuf;
char *tobuf;
int len;
```

DESCRIPTION

The dbcopy library routine provides a binary copy of data from one location in memory to another. The first argument (**frombuf**) is the address of the source buffer. The second argument (**tobuf** determines the location to copy to (destination buffer), and the third argument (**len**) specifies the number of bytes to be copied.

SEE ALSO

dbzero(3I)

RETURN VALUE

This function returns an undefined value. This value should not be tested.

BUGS

This function returns an undefined value. This value should never be tested.

NAME

dbzero - zero fill a buffer

SYNOPSIS

int dbzero(buf, buflen)
char *buf;
int buflen;

DESCRIPTION

The dbzero library routine provides a standard mechanism for zero filling a given buffer of given length. The fisrt argument is the address of the buffer, and the second argument specifies the number of bytes that are to be zero filled.

SEE ALSO

dbcopy(3I)

RETURN VALUE

This library routine always returns an undefined value, but never fails.

BUGS

This library routine returns an undefined value, no test on the value should be made.

NAME

dg_get_next_hop - set next node in user datagram structure

SYNOPSIS

```
#include "dnet.h"

int dg_get_next_hop(udg)
struct udg *udg;
```

DESCRIPTION

This routine will take the values passed in the user datagram structure and will determine the "next hop value" for that datagram. The value of the next hop will be placed in the next.net, next.host, and next.proc fields of the udg structure. The values placed in the next node fields will differ slightly according to wether the next hop is a process on the existing machine, or is the address of another host on a network directly linked to the current machine.

The value of proc always represents the process to send the message containing the datagram to on the current machine. In the case of a datagram arriving at the destination, this represents a user processes bound to process name and may be looked up in the ADGUT. The net and host entries will be set to the same value in the destination node.

In the case of a datagram arriving at a gateway, the process name set represents the bound to process name of the per protocol DGS component that runs the network over which the next hop host is connected to. The net and host names represent the place that the per protocol DGS component is to send the datagram. The net name is required as the DGS component may be responsible for more than one network of a given type.

RETURN VALUE

This routine will return a value of 0 on success and a -1 when an error condition exists.

ERRORS

The call fails if:

[D_NOPATH] The network passed in the user datagram structure could not be resolved in the current host's routing table.

CAVEATS

This routine is defined internally within the dgms component and therefore is inaccessable to any other module.

NAME

disassemble - disassemble a "datagram" for connection services

SYNOPSIS

```
#include "dnet.h"

void disassemble(buf, dg)
char *buf;
struct datagram *dg;
```

DESCRIPTION

This user library routine (used only with the connection oriented services) disassembles a datagram created by one of the following user library routines:

- dn_makedg(3U)
- dn_makepvc(3U)
- dn_makesignal(3U)

The datagram is disassembled into a datagram structure of the following form:

```
struct datagram
{
        short dglen;
        int stream;
        short type;
        short numhops;
        short pathlen;
        char *path;
        char *msg;
};
```

SEE ALSO

dn_makedg(3U), dn_makepvc(3U), dn_makesignal(3U)

NAME

dn_alloc - dynamically allocate memory for dnet structures

SYNOPSIS

```
#include "dnet.h"

char *dn_alloc(s_token, c_token, size, addr)
int s_token;
int c_token;
unsigned *size;
char *addr;
```

DESCRIPTION

The dn_alloc internal library routine (should be implemented for the user library also) dynamically allocates memory for the dnet structures to be used by programs. These routines not only encourage the efficient usage of memory, but also provide for portability of programs if the definition of the structure is modified. If these routines are used, then the template of the structre should not be redefined, and fields should be referenced through the field names provided in the system definition of the structure.

The following structures may be allocated using this routines:

DGMS_MSG This will allocate space that may be accessed through the dgms_msg structure.

DN_UDG This will allocate space that may be accessed through the udg structure.

DN_SVMSG This structure token is only valid and compiled on a Unix System V and will result in a D_BADARG error condition if use is attempted on any other system. This allocates space necessary for the msgbuf structure used in System V message queues.

The c_token parameter must specify one of the following command tokens:

DN_ALLOC This is used to initially allocate the structure. The addr field is ignored.

DN_REALLOC This is used to reallocate the size of an existing structure allocated using the DN_ALLOC command. The addr field must reference the address of a valid structure allocated using the DN_ALLOC command.

DN_DALLOC This command is used to deallocate, or free up the space allocated for the structure, after the structure is of no use. As the amount of dynamically allocated memory increases, the efficiency at which more memory is allocated decreases.

The size parameter is a pointer to an unsigned value. This value is read by dn_alloc to determine the requested size of the buffer field within the structure being allocated. If the value is zero, then dn_alloc will allocate the maximum allowable buffer for that particular structure. The dn_alloc routine will return in the location specified by size the size of the entire allocated structure. The size of the header may be determined by subtracting the number of requested buffer bytes (if non-zero) from the value set after the dn_alloc call. If the size is not initialized to a valid value, the program will behave unpredictably.

The addr parameter is only meaningful when used with the DN_REALLOC or DN_DALLOC commond token. In these cases the address should be the location in memory of a dnet structure previuosly allocated with dn_alloc.

RETURN VALUE

The routine will either return the memory location of the newly allocated structure, or a NULL value indicating an error.

The DN_DALLOC command will always return a NULL pointer.

ERRORS

The call fails if:

[D_SYSERR] A system error has ocurred, check the errno variable to determine what the system error was.

[D_BADARG] An unknown structure token was passed.

[D_BADARG] An unknown command token was passed.

[D_BADARG] The command token was either DN_REALLOC , or DN_DALLOC and the addr field was 0.

[D_MSGTB] The size argument passed with the DGMS_MSG or DN_SVMSG structure token exceeded the maximum allowable size for that structure.

[D_DGTB] The size argument passed with the DN_UDG structure token would exceed the maximum allowable datagram size.

BUGS

This call is implemented on top of the malloc library routines which are ambiguos as to the source of error. Therefor, the dn_alloc routine may incorrectly report a system error when one has not actually occurred.

NAME

dn_initperm - Establish and bind an endpoint for communication

SYNOPSIS

int tcp_initperm(service, backlog)
char *service;
int backlog;

int decnet_initperm(service, backlog, pauxchan)
char *service;
int backlog;
int *pauxchan;

DESCRIPTION

The dn_initperm routines establishes an endpoint for communication over either a TCP/IPC or DECnet provider, binds to the port number specified by service, and specifies that up to backlog connection requests may be outstanding on the established endpoint. In the decnet_initperm routine, the pauxchan points to the location where the file descriptor will be placed for the mailbox associated with the network channel. This is needed to handle multiple inbound requests on VMS.

This call is used to merely set up the endpoint and will not block waiting for a connection request.

The service argument is a character string that has either been defined as being a well known service (in /etc/services on UNIX machines) or is an ASCII representation of an integer value, in which case the value will be used directly as the TCP port to bind to.

SEE ALSO

dn_initperm(3U)

RETURN VALUE

The call returns a valid file descriptor to the endpoint on success or a -1 to indicate an error.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred, check the global variable errno (on UNIX machines) to determine the cause. (UNIX ONLY)

[D_INTR] A signal was caught while attempting to establish the endpoint. No endpoint will be established in this case. (UNIX ONLY)

[D_NODNETSRV] The service name specified could not be found in the definition of servers (/etc/services on UNIX). (UNIX ONLY)

BUGS

The decnet_initperm routine does not currently set any indication for cause of error. The standard VMS error reporting routines should be consulted in when using this routine.

NAME

dn_makedg - assemble a DG_CALLBACK datagram

SYNOPSIS

```
void dn_makedg(buf, channel, numhops, path, msg)
char *buf;
int channel;
int numhops;
char *path;
char *msg;
```

DESCRIPTION

This internal library routine assembles a DG_CALLBACK datagram, used exclusively by the connection oriented service, given the channel number, the number of hops, the path, and message. The contents of the assembled "datagram" are placed into the buf buffer. The assembled datagram resembles:

dg_len | channel | type = DG_CALLBACK | numhops | pathlen | path | msg

The path element is composed of the following. (Sometimes the next and destination hops are the same so the three next elements are eliminated):

thisnet | thishost | thisproc | nextnet | nexthost | nextproc | destnet | desthost | destproc

SEE ALSO

dn_makepvc(3I), dn_makesignal(3I)

NAME

dn_makepvc - assemble a DG_STREAM datagram

SYNOPSIS

```
void dn_makepvc(buf, channel, msg)
char *buf;
int channel;
char *msg;
```

DESCRIPTION

This internal library routine creates a DG_STREAM "datagram" (used only by the connection oriented services) given a channel and message. The assembled "datagram" is placed into the buffer (buf). The datagram looks similar to:

dg_len | channel | type = DG_STREAM | msg

SEE ALSO

dn_makesignal(3I), dn_makedg(3I)

NAME

dn_makesignal - make a DG_SIGNAL datagram

SYNOPSIS

```
void dn_makesignal(buf, channel, msg)
char *buf;
int channel;
char *msg;
```

DESCRIPTION

This internal library routine assembles a DG_SIGNAL "datagram" (used only by the connection oriented services) given a channel and message. The assembled "datagram" is placed into the buffer (buf). The assembled "datagram" looks similar to the following:

dg_len | channel | type = DG_SIGNAL | msg

NOTE: For now, this is identical to dn_makepvc(3I) except that the datagram type is DG_SIGNAL. Eventually, this should assemble something that looks more like a DG_DATAGRAM datagram.

SEE ALSO

dn_makepvc(3I), dn_makedg(3I)

NAME

dnet_error - print dnet stack dump and error description

SYNOPSIS

#include "dnet.h"

void dnet_error(user_message)
char *user_message;

DESCRIPTION

The dnet_error library routine prints out a dnet stack dump and a descriptive error message about the dnet_error that just occurred. If the dnet error indicates a system error, then a descriptive message of the system errror which just occurred will also be displayed.

On top of the error display and stack dump, the message pointed to by the first argument (**user_message**) will be displayed.

BUGS

The descriptive error messages being written are dependant upoon the underlying services setting the **dnet_errno** variable (see dnet_errno.h). In the connection services and on the VMS machines, this variable is not reliably set.

NAME

get_firsthop - get source node description from path string

SYNOPSIS

```
#include "dnet.h"

int get_firsthop(path, firsthop)
char *path;       /* Returned by get_path(3I) */
HOPFIELD *firsthop;/fP
```

DESCRIPTION

The get_firsthop routine will set the value of firsthop to the source node description according to the values in the path string. The get_path(3I) routine may be used to extract routing information, which can then be broken out by this routine, get_nexthop(3I), and get_lasthop(3I).

SEE ALSO

get_path(3I), get_nexthop(3I), get_lasthop(3I)

RETURN VALUE

The return value of get_firsthop is undefined.

BUGS

The return value of this routine is undefined and should be ignored.

NAME

get_lasthop - get destination node from path string

SYNOPSIS

```
/f3#include "dnet.h"

int get_lasthop(path, numhops, desthop)
char *path;
int numhops;
HOPFIELD *desthop;
```

DESCRIPTION

This routine extracts the destination node string from the path string. The path string can be set using the get_path(3I) routine.

If numhops is a non zero value, then this routine will grab the destination node description from the third section of the path string. If numhops is a zero value, then it is assumed that the destination node is being determined on the destination machine. The path string will then only contain two sections, and the destination node description from the second section of the path string.

SEE ALSO

get_path(3I), get_firsthop(3I), get_nexthop(3I)

RETURN VALUE

The return value is undefined for this routine.

BUGS

This routine currently returns an undefined integer value. It should be ignored.

NAME

get_nexthop - get next node description from path string

SYNOPSIS

```
#include "dnet.h"

int get_nexthop(path, nexthop)
char *path;
HOPFIELD *nexthop;
```

DESCRIPTION

This routine will extract the next node description string from the path string. The path string is set by the get_path(3I) routine.

This routine along with get_firsthop(3I) and get_lasthop(3I) make up a set of routines for extracting node descriptions from the path string. Because the path string may vary depending upon the machine it is on, these routines should be used to extract the node descriptions rather than accessing the path string directly.

SEE ALSO

get_path(3I), get_firsthop(3I), get_lasthop(3I)

RETURN VALUE

This routine currently returns an undefined integer.

BUGS

This routine currently returns an undefined integer value. It should be ignored.

NAME

get_path - low level dnet routing function

SYNOPSIS

```
#include "dnet.h"

char *get_path(source, destnet, desthost, destproc, numhops)
struct nethost_entry *source;
char *destnet;
char *desthost;
char *destproc;
int *numhops;
```

DESCRIPTION

This internal library routine provides the low level dnet routing service for dnet components. Given the source and destination networks, hosts, and processes, this routine determines wherre the next hop is. If the source and destination networks are the same, a two part path is assembled, aconsisting of the following:

thisnet|thishost|thisproc|destnet|desthost|destproc

In that case, a value of 0 s placed in numhops.

If the source and destination networks are different, the router looks in the network routing table (loaded into memory by dn_init(3I)) for an entry wherre the source and destination networks match the source and destination networks passed to this routine. If a match is found, a path is returned that looks similar to:

thisnet|thishost|thisproc|nextnet|nexthost|nextproc|destnet|desthost|destproc

A value of 1 is placed into numhops to indicate this type of path.

SEE ALSO

get_firsthop(3I), get_nexthop(3I), get_lasthop(3I)

RETURN VALUE

A valid character pointer is returned on success, and a NULL pointer is returned to indicate an error.

NAME

ipcclose - close an ipc mechanism

SYNOPSIS

int ipcclose(ipcid)
int ipcid;

DESCRIPTION

The ipcclose internal library function removes the calling process's access to the ipc mechanism identified by ipcid. Any later access to that ipcid will be invalid.

If the mechanism being closed was accessed by the user using the D_BIND flag in the ipcget(3I) routine, then the mechanism will be removed from the system. If the D_BIND flag was not specified, then the mechanism will remain in the system until the binding peer issues the ipcclose(3I) call. Even if the mechanism remains intact, the user will still not be able to access after the ipcclose.

SEE ALSO

ipcget(3I)

RETURN VALUE

Upon successful completion, the function will return a value of 0. If an error occurred, then the function will return a value of -1 and will set the variable dnet_errno to indicate the error condition.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

[D_BADARG] The ipcid passed was invalid.

[D_EPERM] Write permission is denied on the ipc directory, or search permission to the ipc directory is denied. This indicates that permissions have been changed since the time that ipcget was called.

[D_NODNET] The ipc directory no longer exists.

[D_NOEXIST] The ipc mechanism has already been removed. This usually means someone has manually removed the file node.

CAVEATS

If ipcid is valid, the ipc mechanism will be closed by this routine even if an error occurrs.

NAME

ipcget - establish and/or gain access to an IPC mechanism

SYNOPSIS

```
#include "dnet.h"

int ipcget(name, flags)
struct dnet_ipcname *name;
int flags;
```

DESCRIPTION

The ipcget library routine is used to establish and/or gain access to a mechanism for interprocess communiction.

The following is the template definition for the dnet_ipcname structure:

```
struct dnet_ipcname
{
        char name[D_MAXPATHNAME];
        unsigned msgsize;
        unsigned mqueuesize;
};
```

The name field represents a string value that will be used to determine peers in a conversation. The name chosen may not contain the forward slash character.

The msgsize field represents an attempt at negotiation between the user process and dnet for determining the maximum size of message that may be passed through. If the icpget call succeeds, then dnet guarantees that messages of that size or smaller will not be truncated. The ipcget call will fail if the underlying IPC mechansisms are not capable of handling a message of the size requested.

The mqueuesize argument is used to request that dnet attempt to allocate enough space to allow mqueuesize number of messages of msgsize to be sent to the queue without ever blocking. This is infeasible in most environments because of sharing of buffering space with other processes, but can be used to warn dnet of the expected activity for a particular user. Future releases of dnet may actually take back allocated space if it is needed for other users.

An integer (ipcid) will be returned on successful completion which must be used in future calls to the established IPC mechanism.

The following flags may be set:

D_BIND	Specifies that name is to be used to identify what incoming datagrams are to be received at this endpoint. Only one process is allowed to bind to a given name at a time. Either the D_BIND, or the D_CONNECT flag must be specified.
D_CONNECT	Specifies the address (name) to which all datagrams leaving via this IPC mechanism are to be sent. This flag is mutually exclusive with respect to D_BIND. At least one of these mutually exclusive flags (D_CONNECT, D_BIND) must be specified in the flags parameter.
D_GLOBAL	This flag is only meaningful when used in a VMS environment in combination with the D_BIND flag. The effect of this flag is to advertise the name of the ipc mechanism in the system table, rather than just the job table. This flag will cause the call to fail if the calling process does not have SYSNAM privilege.

SEE ALSO

ipcsnd(3I), ipcrcv(3I), ipcclose(3I)

RETURN VALUE

The function call will return a positive number representing a valid ipcid, or will return a -1 indicating an error and the external variable dnet_errno will be set to the error code.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_BADNM]	The name was either determined to have a length of zero, or the length of the name was longer than the system imposed maximum (see D_MAXPNAME in dg.h). All names are assumed to be null terminated string values.
[D_BADMN]	The name contained the forward slash (/) character.
[D_BADARG]	Both the D_BIND and D_CONNECT flags were specified.
[D_AEXIST]	The D_BIND flag was set and another process was already bound to the address in name->name.
[D_NOEXIST]	The D_CONNECT flag was set and there was no process bound to the given name.
[D_NOSRSC]	There are currently not enough system resources available to provide for another IPC mechanism at this time. The call may succeed at a later time.
[D_NOSRSC]	You have too many ipc mechanisms active. You will need to perform an ipcclose(3I) before you issue another ipcget(3I).
[D_QUOTA]	Your process has the maximum number of file descriptors already in use.
[D_NODNET]	The error that occurred would indicate that the proper dnet components were not started up, or were not started up properly. One or more of the followin indications were found: • A component of the dnet assembled absolute pathname for the IPC mechanism was determined to not be a directory. This is indicative of absence of the dnet temporary directory from this machines file hierarchy. • If the current system is Unix System V, the error may have resulted from the dnet message queue(s) not existing.
[D_EPERM]	Search permission of a component of the dnet temporary directory was denied the calling process, or write permission to the dnet temporary directory itself was denied.
[D_EPERM]	If the current system is Unix System V, this error may have occurred from lack of permission to the message queue(s).
[D_EPERM]	If the current system is VMS, then the user may not have permission to create mailboxes.
[D_INTR]	The system call was interrupted by the receipt of a signal before it could be completed.

BUGS

None of the size fields within the dnet_ipcname structure are currently supported or checked. This was provided for VMS implementations where the IPC queueing space is explicately allocated for each mechanism.

NAME

ipcrcv - receive an ipc message

SYNOPSIS

```
#include "dnet.h"

int ipcrcv(ipcid, msg, msglen, flag)
int ipcid;
char *msg;
int msglen;
int flag;
```

DESCRIPTION

The ipcid argument is the integer handle returned from a successful ipcget routine.

The ipcrcv function call allows a process to receive an incoming message on the specified ipcid. A blocking read is performed unless the D_NOWAIT flag has been set.

The value in msg is an address of a character array where the message will be placed. No more than msglen characters will be read. Any extra characters will be truncated.

SEE ALSO

ipcsnd(3I)

RETURN VALUE

Upon successful completion, the function will return a value representing the number of characters received. If an error occurred, the value returned will be -1 and the variable dnet_errno will be set to indicate the specific error condition.

ERRORS

The call fails if:

[D_SYSERR]	A system error has occurred. Check the global variable errno.
[D_BADARG]	The ipcid passed was zero or did not reference a valid dnet ipc mechanism.
[D_BADARG]	The specified buffer length was less than one.
[D_NOMSG]	The D_NOWAIT flag was set and no messages were waiting to be read.
[D_EPERM]	Read permission on the underlying IPC mechanism was denied to the calling user.
[D_NOEXIST]	The peer reset it's connection. The ipcrcv routine will issue an ipcclose(3I) on this ipcid to invalidate it for you. On System V machines, this actually means that the dgms component reset the entire ipc medium.
[D_INTR]	A signal was caught while attempting to read from the ipc mechanism. No message was read in.

BUGS

The D_NOWAIT flag requires a system call after receiving a message in the BSD environment. This opens up the possibility of a signal being posted after a successful read. This situation will cause a D_INTR error to be specified and the ipcrcv call will appear to fail. If the D_INTR message is set, check to see if the message was actually read, and if so, reissue another non-

blocking read to reset the socket endpoint properly.

NAME

ipcsnd - send a message via a dnet IPC mechanism

SYNOPSIS

int ipcsnd(ipcid, msg, msglen, flags)
int ipcid;
char *msg;
int msglen;
int flags;

DESCRIPTION

The ipcsnd function call allows a process to send a message out an IPC mechanism created with the ipcget library routine.

The only flag value currently supported is the D_NOWAIT flag which will insure that the calling procedure will not block on back pressure from the underlying IPC mechanism.

SEE ALSO

ipcrcv(3I)

RETURN VALUE

Upon successful completion, the ipcsnd function call will return a value of 0. If an error occurred, a value of -1 will be returned and the dnet_errno variable will be set to indicate the error code.

ERRORS

The call fails if:

[D_SYSERR] A system error has occurred. Check the global variable errno.

[D_WOULDBLOCK] The D_NOWAIT flag was set and sending the message at this time would cause the process to block waiting for the underlying mechanism to release back pressure.

[D_BADARG] The ipcid value passed was either zero, was a negative number, or did not represent a valid dnet IPC mechanism.

[D_BADARG] The ipcid passed represents an IPC mechanism created with D_BIND, and therefore cannot be used with ipcsnd.

[D_BADARG] The value of umsglen was determined to be less than one or greater than the maximum allowable message size (D_MAX_IPC_MSG_SIZE in dnet_ipc.h).

[D_NOEXIST] The peer reset it's connection. The ipcsnd routine will issue a ipcclose(3I) for your process.

CAVEATS

Unix System V implementations attempt to dynamically allocate memory space for sending messages when they are called from within a dnet user program. This may result in a system error occurring from temporary lack of memory space which may be available at a later time. The expected results would be that dnet_errno would be set to D_SYSERR, and errno would be set to EAGAIN. The current implementation provides no explicit or guaranteed method for determining this condition.

BUGS

No explicit and guaranteed indication of temporary lack of dynamically allocatable memory space is provided by dnet.

NAME

is_error - print error message if system call return value indicates error

SYNOPSIS

```
int is_error(retval, errmsg)
int retval;
char *errmsg;    /* Message to print if error occurred */
```

DESCRIPTION

This internal library routine is meant to be called after a system call. If the value returned by the system call (retval) indicates an error, the the errmsg is displayed.

NOTE:

This function has probably outlived its usefullness. The original intent was to get a handle on errors returned on the VAX. Some system calls (those implemented by Wollongong) return an error value but fail to set errno so that you can't learn anything by calling perror(). Instead, another external variable, uerrno, was set. This function was needed to get the value of uerrno so we coulde ook it up in the errno.h file manually.

SEE ALSO

dnet_error(3U)

RETURN VALUE

This routine returns a vlue of 0 if retval is not negative, and a 1 if it is.

NAME

load_my_name - determine the name of this host

SYNOPSIS

#include "dnet.h"

int load_my_name()

DESCRIPTION

This internal library routine loads the entry from the myname table into the myname structure. The myname_table array is defined in the dnet.h header file and is of type struct nethost_entry. The nethost_entry structure is defined as follows:

```
struct nethost_entry
{
        char netname[MAXNAMESIZE];
        char hostname[MAXNAMESIZE];
};
```

The load_my_name routine determines these values from the tbls.myname file in the dnet home directory.

If your module contains a main function definition, then the following line must be in your code before the inclusion of dnet.h:

#define MAINPROGRAM

SEE ALSO

dn_init(3U), load_net_table(3I)

RETURN VALUE

This routine returns a zero on success and a -1 on failure.

ERRORS

The call fails if:

[D_NOSYSFILE] The tlbs.myname file could not be found in the dnet home directory, or was in an invalid format.

[D_SYNERR] More than one non-commented entry was found in the tbls.myname file.

NAME

load_net_table - load routing table into memory

SYNOPSIS

```
#include "dnet.h"

int load_net_table()
```

DESCRIPTION

This internal library routine is used to load the current host's routing table into memory for quicker access and use by future routing functions.

The table is loaded into a structure array defined in dnet.h and named net_route_table. The structure type is net_route_entry and is defined as follows:

```
struct net_route_entry
{
        char srcnet[MAXNAMESIZE];
        char destnet[MAXNAMESIZE];
        char gateway[MAXNAMESIZE];
        char dgsproc[MAXNAMESIZE];
};
```

The table is initialized from the tbls.net file in the dnet home directory.

If your module contains a main function definition, then you will need to add the following line above the inclusion of dnet.h:

```
#define MAINPROGRAM
```

SEE ALSO

dn_init(3U), load_my_name(3I)

RETURN VALUE

This routine returns a zero on success and a -1 on failure.

ERRORS

The call fails if:

[D_NOSYSFILE] The tbls.net file was not found in the dnet directory, or read permission was denied.

[D_NOSRSC] The tbls.net file contained more records than were defined for the internal table. Look at the value of MAXTBLSIZE in the dnet.h header file.

[D_SYNERR] A record was found in the tbls.net table that was determined to have the wrong number of fields.

NAME

makeipc - administrative creation of an IPC medium

SYNOPSIS

```
#include "dnet.h"

int makeipc()

int _makeipc(sv_msg_key, ipcdir, flags)
int sv_msg_key;
char *ipcdir;
int flags;
```

DESCRIPTION

These library routines provides for the administrative creation and general access of a dnet IPC medium. The creation of an IPC medium (not to be confused with creation of an IPC mechanism as described in ipcget) allows creation of a private "area" within the means of inter process communication of the operating system. The intention is to avoid collisions with unrelated processes using inter process communication. The creation of a private area differs according to the operating system.

On all UNIX machines, the UNIX filename is supported for addressing a particular IPC mechanism. To facilitate this, an ipc directory is used to place all addresses (only filenames are supported, explicate pathnames will cause an error on ipcget). If, in addition, the machine hosts a System V operating system, a System V message queue is also required. In a VMS environment, these routines are effectively empty functions.

All IPC routines require that the IPC medium be accessed by all processes wishing to use it. In addition, an administrative process needs to create it before any other processes attempt to access it. The flags parameter allows the _makeipc routine to be issued for creation by using D_CREAT | D_EXCL. The makeipc routine calls _makeipc with the flags set in this fashion. If _makeipc is called with only the D_CREAT flag specified, then the _removeipc (or removeipc) will always return successfully without removing the medium. Processes other than the administrative process should attempt to "access" the IPC medium by setting the flag values to 0. The call will fail in this case if the IPC has not been created.

The first two parameters to the _makeipc routine allow the processes to choose the IPC directory and System V message queue key value (only meaningful on a System V machine). These parameters allow for avoidance of collisions with other, unrelated processes using the interprocess communication means for a particular machine. In addition, the proper placing of the IPC directory may also provide additional security inherant within the UNIX filestore.

SEE ALSO

ipcget(3I), removeipc(3I)

RETURN VALUE

Upon successful completion, the function will return the value of the msqid for the System V message queue created, or a 0 in other environments. The msqid, though, is of no use to other IPC routines, since the _makeipc routine makes it available to them transparent to the user.

If an error occurred, then the function will return a value of -1 and will set the variable dnet_errno to indicate the error condition.

NAME

prttime - return a string representing the current time of day

SYNOPSIS

char *prttime()

DESCRIPTION

This internal library routine returns the current time of day in a character string of the form: "time: 12:59:59". Hours, minutes, and seconds are given.

RETURN VALUE

This routine returns a pointer to the character string generated.